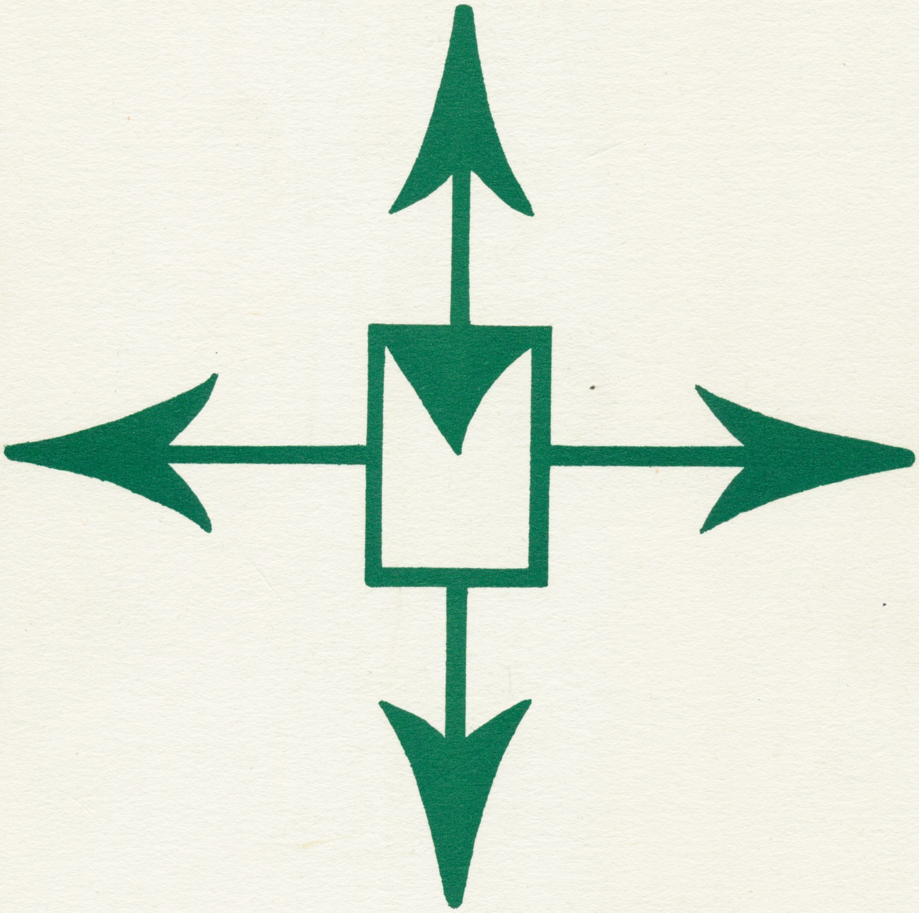
APL V



FIFTH INTERNATIONAL
APL USERS' CONFERENCE

MAY 15-18, 1973

TORONTO, CANADA

PROCEEDINGS
OF THE
FIFTH INTERNATIONAL
APL USERS' CONFERENCE

MAY 15-18, 1973

THEME

APL APPLICATIONS IN EDUCATION
AND BUSINESS

Copyright 1973 by the APL Technical Committee

Printed by Canadian Printco Limited

TORONTO ONTARIO CANADA

FOREWORD

The fifth annual APL Users' Conference was held in Toronto, Ontario on May 15 to 18, 1973 at the Inn on the Park. The conference banquet was held at the nearby Ontario Science Centre. Toronto was chosen to host the conference because of the extensive APL activity in the area. Most universities in Ontario offer APL, and it is also heavily used in neighbouring Quebec and northern New York State.

The theme of the conference was "APL Applications in Education and Business". It was felt that the APL Conference in Copenhagen in August would offer a forum for those whose APL interests were in further enhancements of the language and technological developments. Therefore, the organizers agreed to emphasize the use of APL as an educational and commercial tool. The papers published in these proceedings reflect that emphasis.

The selection of papers and the scheduling of the conference was the responsibility of Mr. Ted Edwards, of Control Data Canada, Limited, who served as Program Chairman.

On the Local Arrangements Committee Mr. Sam Goldfarb, University of Toronto, was responsible for all printing and mailing, and especially for the publication of these proceedings. Mr. Don Kellett, York University, assumed responsibility for the on-line registration system and for organizing the equipment display areas. Mr. Brian Hockey, I. P. Sharp Associates, served as Treasurer and Ms. Mary Kiely served as Secretary and also ably organized the banquet. I am grateful for all the time and ability these people devoted to their responsibilities and have enjoyed my association with them.

The organizers are also grateful to Dr. Ken Iverson for his encouragement and support, and to Dr. Garth Foster, whose guidance and suggestions were invaluable.

We are also grateful to all the contributors of papers, whose enthusiastic support contributed to the success of the conference.

Any profits from the conference will be turned over to the APL Users' Committee to further the publication of "Quote-Quad".

It is hoped that the conference succeeded in furthering the acceptance of APL and that those who attended the conference found the papers and new associations of value.

Larry B. Moore
Local Arrangements Chairman

TABLE OF CONTENTS

H.A. PEELLE
TEACHING CHILDREN THINKING VIA APL 0

K.P. CAROLAN
TEACHING APL 1

D. WATSON
AN INTRODUCTORY APL SELF TEACHING PACKAGE 2

G.H. FOSTER
MOTIVATING ARRAYS IN TEACHING APL 3

E.E. MCDONNELL
THE VARIETY OF DEFINITIONS OF A SIMPLE FUNCTION 4

J.R. CLARK, R. MERCER
TECHNIQUES IN PROGRAMMING CAI WITH APL 5

S. CHARMONMAN, D.K. REED
A TABLE DRIVEN PROGRAM TO PLAY A GAME OF LOGICAL DEDUCTION 6

L.N. DAVIS, J. ESKINAZI, D.J. MACERO
COMMUNICATION, COMPUTERS, AND THE CLASSROOM 7

G. BARTOLI, L. BARTOLO, V. SPADAVECCHIA
A SIMULATOR FOR USE OF AN APL COMPUTER IN SCIENCE TEACHING 8

J.R. SAVARY
APL AS AN AID TO THE TEACHING OF UNDERGRADUATE ECONOMICS 9

K.E. IVERSON, INVITED SPEAKER
APL AS AN ANALYTIC NOTATION 10

G. WILHELMI
FORMAL DIFFERENTIATION USING APL 11

P. PENFIELD
PROPOSED NOTATION AND IMPLEMENTATION FOR DERIVATIVES IN APL 12

M.M. HALPERN
PERMUTATIONS 13

J.W. ROUGH
XEROX FINANCIAL PLANNING AND CONTROL SYSTEM IN APL 14

D. FLOWER
THE BROWNBOX INFORMATION RETRIEVAL SYSTEM 15

TABLE OF CONTENTS

H.M. COURTNEY, C.M. CHENEY
AN APPLICATION OF THE BINOMIAL DISTRIBUTION TO SALES PLANNING 16

W.R. TINGA
ADMINISTRATIVE DATAFILE HANDLING VIA APL 17

D. VASKEVITCH
APL FOR INFORMATION MANAGEMENT AND CAI 18

D.G. MOFFAT, M.A. JENKINS
APL/11 - A POOR MAN'S APL 19

H.J. SAAL
SIMULATING COMPUTERS IN SYMBOLIC ASSEMBLER NOTATION 20

P.H. RUSSEL, L. WAGSHAL, J.C. WIGINTON
APPLICATION OF A HYBRID (TWO-MODE) PROCESSING SYSTEM 21

I.M. BRAUDE, M.A. JENKINS
AN APL TRANSLATOR FOR THE B6700 22

"Teaching Children Thinking Via APL"

by

Howard A. Peelle

University of Massachusetts

April 1973

Abstract

Proposed is a new approach to teaching children topics which require thinking. Using A Programming Language (APL), short quickly digestable computer programs are given to elementary school children for their direct viewing. Each program embodies a concept, procedure, or interrelationship, and--written simply and comprehensibly--serves to reveal or clarify understandings. The definition of such a program makes a procedure visible in concrete, explicit terms. This is more a "glass box" than a "black box" approach; that is, by viewing the APL program, key concepts can become transparent to the student.

The "glass box" approach to teaching children thinking represents an extension of previous work by three others. Paul Berry first espoused open use of APL glass boxes as a strategy for teaching. (cf. "APL and Insight: A Strategy for Teaching," P. Berry, IBM). Seymour Papert has stressed the importance of studying procedures actively with a computer programming language as a conceptual framework (cf. "Teaching Children Thinking," S. Papert, LOGO Memo #2, M.I.T.). Kenneth Iverson has persistently advocated simplicity and generality in the use of APL for exposing fundamentals in a variety of mathematical and scientific disciplines (cf. "APL in Exposition", K. E. Iverson, IBM Tech. Report #320-3010).

In contrast to conventional computer assisted instruction (CAI), the glass box approach seeks to give the student control over his own learning. Making the full power of the computer available via programming is 180° from

the kind of CAI characterized by lock-step tutorial, drill-and-test instructional sequences. To children who typically have been powerless in authoritarian educational systems, use of the computer in this way potentially opens up new worlds of learning--active learning, learning with power.

With "glass box" computer programs, children can proceed to think and to learn during several complementary activities. Specifically, they can: (1) examine the program's definition, using logic or intuition, (2) execute the program on a computer, scrutinizing its behavior, (3) experiment with different uses of the program, trying special arguments, etc. (4) modify or expand the program, (5) generalize the program, (6) invent new or related programs, and (7) discuss intellectual implications with teachers or peers.

The glass box program is also expository; that is, it 'speaks' to reader, exposing and clarifying concepts and is, itself, a pedagogical agent. The ideal glass box program is:

Simple
Brief
Comprehensible
Flexible
Adaptable
General
Elegant
Provocative

To illustrate this approach, several sample APL programs are presented with accompanying suggestions for actual use in a teaching-learning setting. Examples are special topics chosen from the following areas:

Cybernetics
Psychology
Mathematics
Computer Art
Linguistics
Statistics
Ecology
Cryptography

Complete paper available from the Author.

TEACHING APL

Kevin P. Carolan
Marist College

INTRODUCTION

Many introductory texts in Computer Science are divided into two parts. The first part is ordinarily a general introduction to computing including a description of the hardware and software aspects of computing, Boolean Algebra, and a history of computers. The second part of the text is devoted to the consideration of a programming language. I have been teaching an Introduction to Computing course at Marist College for three semesters, using APL as the programming language. The college is an institutional member of SECOS (Shared Educational Computer Systems) and has three IBM 2741 Communications terminals on campus. Since there were no introductory texts available which featured instruction in APL as the programming language, I decided to use the audio-tutorial method as the mode of language instruction.

AUDIO-TUTORIAL METHOD

Two one-hour audio cassette tapes were prepared to introduce the students to the fundamentals of APL. The tapes were designed to be used by the student as he worked at a terminal. The method, of course, permits the student to learn at his own pace. In composing these tapes, an effort was made to consider only "non-mathematical" functions or functions that require little mathematical training.

The sequence of topics was:

Tape I - Side A
- Sign-on procedure
- Correcting mistakes
- Order of operations
- Two modes of operation
- Scalars and vectors
- Reduction
- Catenation
- Assignment or specification
- Public libraries
- Sign-off procedure

Tape I - Side B
  Dyadic and monadic functions
  Dimension
  Indexing
  Grade up and grade down
  Writing functions

Tape II - Side A
  Literal characters
  Restructuring vectors into matrices
  Quad for numerical and literal input

Tape II - Side B
  Global and local variables
  Header of a program
  Logical functions
  Relational functions
  Compression
  Branching

The students were expected to complete the two tapes during the first six weeks of a fifteen-week semester. At the end of the six-week period, they were polled in order to determine what level of achievement they thought they had reached. The questions and the results were:

I was able to perform the mechanical procedures of signing-on, loading workspaces, correcting errors, etc.:

	Replies	%
1. Not at all	0	0
2. With difficulty	1	3
3. Reasonably well	19	58
4. Very well	13	39
5. None of these	0	0

My knowledge of the APL functions that were discussed is:

1. Very poor	0	0
2. Poor	2	6
3. Fair	12	36
4. Good	17	52
5. Very good	2	6

When it came to writing functions or programs, I:

		Replies	%
1.	Was completely lost	2	6
2.	Could copy the program but did not understand what I was doing	9	27
3.	Could copy and had some understanding of what I was doing	18	55
4.	Can write a simple program	4	12
5.	None of these	0	0

The length of the tapes (2 hours) is:

1.	Too little	2	6
2.	About right	24	73
3.	Too much	6	18
4.	None of these	1	3

The amount of time that I needed to complete the two tapes was:

1.	2 1/2 - 3 hours	9	27
2.	3 - 3 1/2 hours	9	27
3.	3 1/2 - 4 hours	8	24
4.	More than 4 hours	6	18
5.	None of these	1	3

I thought that the tapes were:

1.	Not effective	0	0
2.	Somewhat effective	6	18
3.	Effective	17	52
4.	Very effective	10	30
5.	None of these	0	0

Although 43 students were enrolled in the course, 10 students did not use the tapes for various reasons and their responses are not included in the results. The students were then tested on the material covered in the tapes. The results on that examination confirmed the students' self-evaluation of their knowledge of APL.

To supplement the introductory content of the audio tapes, a glossary of additional APL functions and terms was prepared and saved in two workspaces. A student who had taken the course the previous year wrote the description of the functions and terms. Two advantages to storing the glossary in a workspace are:

(1) Revision of the glossary is an easy procedure;

(2) A terminal user can look up a function or term simply by copying its description into his active workspace.

At the end of the semester, the students were polled again to discover their reaction to the glossary. The response to the following two questions indicates that the lack of a formal text was not a major obstacle for the majority of the students and that the glossary was quite adequate for their needs.

The lack of a text was:

Severe handicap	1	2	3	4	5	No handicap at all
Replies	1	14	4	6	18	
%	2	33	9	14	42	

The glossary of APL terms was:

Totally inadequate	1	2	3	4	5	Very adequate
Replies	0	0	6	14	23	
%	0	0	14	33	53	

SUMMARY AND IMPLICATIONS

Most educators would agree that the more an educational curriculum can adapt in a unique fashion to individual learners, the better the chance is of providing the student with a successful learning experience. The audio-tutorial method seems to be a very natural way to adapt to the individuals who are learning an interactive programming language. A student who is having difficulty with a particular item can replay it until he has learned it. Since an audio tape is easy to prepare, instructors can have different tapes for various purposes and groups. An instructor can also devote more time to students who have specific programming needs since he does not have to spend time explaining elementary functions and operations. In addition to being responsive to the individual learner, the audio-tutorial method is economical. The equipment is relatively inexpensive and the time of both the students and the instructor is used more efficiently and effectively.

Even more important than the efficiency and effectiveness of the audio-tutorial approach in teaching a conversational programming language, is its educational implications. Marshall McLuhan has made the following observation, "What the computer means in education is this. As information speeds up, information levels rise in all areas of mind and society, and the result is that any subject of knowledge becomes substitutable for any other subject. That is to say, any and all curricula are obsolete with regard to subject matter. All that remains to study are the media themselves, as forms, as modes ever creating new assumptions and hence new objectives."[1] Certainly, the best way to study the computer as a mode which ever creates new assumptions and new objectives is to introduce students to the computer and then allow them to discover the technique of discovery.

---

[1] M. McLuhan, "Is it Natural that One Medium Should Appropriate and Exploit Another?" McLuhan: Hot & Cool (New York: The New American Library, Inc., 1969) p. 159.

An Introductory APL
Self Teaching Package

During 1970 an APL self teaching C.A.I. package was developed at Lakehead University. The aims of the package were:

1) Salesmanship
2) To give the new user confidence and experience in the use of a 2741 terminal and the APL system
3) To be foolproof over its early stages
4) To teach many of the aspects of APL which could not be taught without an APL terminal.

The package proved popular and it has become the only way that APL is introduced at Lakehead University. Instructors can use APL in their courses without having to dedicate lecture time to APL. To the Computer Centre it has meant that APL runs itself without any academic support.

Complete paper available from the Author.

Motivating Arrays in Teaching *APL*

Garth H. Foster
Electrical and Computer Engineering
Syracuse University
Syracuse, New York, 13210 USA

1. INTRODUCTION

It is self evident that the mode of teaching *APL* has been strongly influenced by the published materials which have been made available to educators. The first of the User's Manuals by Falkoff and Iverson (1968,1970) was followed by texts by Gilman and Rose(1970),Katzan(1970,1971), Prager(1971), and more recently, Grey(1973). All of these works set forth a reasonably complete presentation of the language, at the level of*APL*\360. Since these texts are intended to be tools for learning the language, the contents are not best arranged so as provide a reference to the language. That function has been fulfilled by Pakin (1968,1972).

The usual mode of presentation of *APL* in most of these texts is to: (1) introduce the terminal, its character set, operation and protocol for communication with the system; (2) introduce a few of the more familiar scalar dyadic functions and use these to develop assignment of values to variable names and the evaluation of expressions without full parenthesization; (3) present vectors and reduction early along with function definition. (4) More functions are introduced with examples appropriate to the new material. (5) Finally, higher dimensioned arrays are brought into the view of the student and exposure is given to the more esoteric array functions. Functions such as reduction, reshape, catenation, rotation, compression, and expansion which were covered earlier are now extended to cover structures dealing with ranks greater than one.

It is not the purpose of this paper to take issue with this approach for these methods have been proven to be highly successful in communicating the concepts of *APL*. We do want to point out that in teaching *APL* greater emphasis can (and should) be placed on two closely related aspects: (1) making an effort to re- turn to the same problem using more complex formulations and (2) placing an increased emphasis on the array.

Texts generally fall short in both of these areas because, on the one hand, there appears to be a compelling force when introducing new material to write about entirely new applications for the features just covered. This is not altogether bad because it does give added scope to the text and the use of the language. It does, however, place added responsibility on the shoulders of the instructor to provide a setting where a problem can be seen from a number of contexts as the student's understanding of the language increases. In the second case, the student arrives at array material nearly at the end of the journey through *APL* with the result that there is little or no reinforcement of learning in that area. The hope of finding suitable educational materials in application of *APL* is soon dashed on the realities of the publishing industry. Most books dealing with applications do not concern themselves with *APL*; in fact they have titles something like: ENGINEERING APPLICATIONS WITH FORTRAN IV or BUSINESS APPLICATIONS WITH BASIC and in these contexts the use of the array as a structure for problem solution is rare.

The following material is intended to illustrate the spirit of the method rather than to be intended to be a syllabus for a course.

2. ITERATION IN THE LEARNING OF PROGRAM CONCEPTS

Consider the programming task of generating the first *N* terms of the Fibonacci sequence. This is a problem which is short and straightforward in any programming language, requiring only assignment, addition with integer atithmetic, and branching. A BASIC approach to this problem is shown as function *FIB*1 in Figure 1a. By basic we mean that it is written at about the level of BASIC, and by printing the values as they are generated, minimal space required for data storage. Usually, we want to retain the sequence and a vector of values is used to this end, as shown in *FIB*2, Figure 1b. This probram has done little to trade scalars for vectors however. We are still at the level of BASIC or FORTRAN except that we have exploited the convenience of catenation and the absence of dimensioning to not have to upper bound the number of elements to be generated. It is still the case that we index to scalar values. Figure 1c, showing *FIB*3 indicates how we may use a vector as an index, combined with reduction, to accomplish the same purpose. It is now only a minor alteration to replace indexing with take to give the function, *FIB*4 of Figure 1d. At this point we have a program which loops explicitly to compute the first *N* values of the Fibonacci series. If we want only the *N*th element it is necessary to do a ¯1↑*FIB*4 *N*. The current sequence generator does provide a tool for further investigation however. The student can be asked to consider (*FIB*4 *N*)+(1↓*FIB*4 *N*+1) to discover that the result equals 2↓*FIB*4 *N*+2. If the mathematical level of the student warrants, the development of the difference equation and its solution can follow leading to *FIB*5 of Figure 1e. Should the class not have the background to follow the difference equation development then the behavior of (1↓*FIB*4 *N*+1)÷*FIB*4 *N* for say *N*←25 can be used to empirically discover the the ratio between successive terms of the sequence approaches Δ of *FIB*5 . Thus, by examining how closely (÷5*.5)×Δ*ι*N* approximates *FIB*4 *N* or by examining how

⍙←1-Δ contibutes to the series by studying the behavior of (÷5*.5)×⍙*⍳*N*, the student can arrive at *FIB*6 of Figure 1f.

In the last two cases we have closed forms for the Fibonacci series and we may calculate (to within the capabilities of the implementation) the *N*th term, the first *N* terms of some subset of the first *K* terms. One should not lead the student to believe that this mode of presentation will always lead to closed forms. I believe that it will be true that more interesting and robust program forms will be found by looking for them while providing the learner the opportunity to explore alternate methods of coding the problem.

```
      ∇ FIB1 N;A;B;C;K
 [1]     A←1
 [2]     B←1
 [3]     A
 [4]     B
 [5]     K←2
 [6]    MORE:C←A+B
 [7]     C
 [8]     A←B
 [9]     B←C
 [10]    →(N≥K←K+1)/MORE
      ∇
```

Fig. 1a

```
      ∇ Z←FIB2 N
 [1]     Z← 1 1
 [2]     →(N≤ρZ)/0
 [3]     Z←Z,Z[ρZ]+Z[¯1+ρZ]
 [4]     →2
      ∇
```

Fig. 1b

```
      ∇ Z←FIB3 N
 [1]     Z← 1 1
 [2]     Z←Z,+/Z[¯1 0 +ρZ]
 [3]     →(N>ρZ)/2
      ∇
```

Fig. 1c

```
      ∇ Z←FIB4 N
 [1]     Z← 1 1
 [2]     Z←Z,+/¯2↑Z
 [3]     →(N>ρZ)/2
 [4]     Z←N↑Z
      ∇
```

Fig. 1d

```
      ∇ Z←FIB5 N;Δ;⍙
 [1]     Δ←0.5×1+5*0.5
 [2]     ⍙←0.5×1-5*0.5
 [3]     Z←(÷5*0.5)×(Δ*N)-⍙*N
      ∇
```

Fig. 1e

```
      ∇ Z←FIB6 N;Δ
 [1]     Δ←0.5×1+5*0.5
 [2]     Z←⌊0.5+(÷5*0.5)×Δ*N
      ∇
```

Fig. 1f

Figure 1
Fibonacci Functions

3. ARRAY PROBLEMS

Often in dealing with array oriented problems the difficulty in the problem lies with grasping the conception of how the array can be used because the task is one of data structuring and manipulation. A problem where this might occur is to ask for what triples of decimal digits, *A*,*B*,*C* does the following 'strange' law of cancelation: ((10⊥*A*,*B*)÷10⊥*B*,*C*) = *A*÷*C* hold? Thus, when we have 33÷33 we have the same result (trivially) as 3÷3, but less obvious by the 'strange law of cancelation' we have that 16÷64 is the same as 1÷4 after we cancel 6's.

The first problem is to generate vectors, *A*,*B*,*C* or a three rowed (or columned) matrix to hold the appropriate values and then test the 'law' as given to find all of those cases which hold and finally print the result. A possible solution is given in Figure 2 in the function *CANCEL*. It is not claimed to be the most elegant function and the type of optimization presented by sykes(1973) has not been done except to make use of ⍳ as applied to arrays, as that extension was not available when the function was first written.

When this problem was first posed to me it was by someone responsible for FORTRAN education and he felt that this was a good problem to reinforce the concept of looping as he saw it as a program with nested DO loops. The transfer of the problem form one of cyclic data generation and examination to one of generating all the data and eliminating that which is not relevant is central to the array nature of the *APL*formulation. If we seek to nurture the spirit of investigation in the hope that it will lead to new and useful formulations as kolsky (1969) suggests, then the array needs to be further stressed in applications.

```
      ∇ Z←CANCEL
[1]     Z←(, 3 2 1 ⍉Z),[1](,⍉Z),[0.5],Z← 9 9 9 ρ⍳9
[2]     Z←((Z[3;]×10⊥Z[1 2 ;])=Z[1;]×10⊥Z[2 3 ;])/Z
[3]     Z← 1 1 0 0 0 1 1 ⍀'123456789'[Z[1 2 2 3 ;]]
[4]     Z[4;]←'÷'
[5]     Z←⍉Z
      ∇
```

Figure 2
A Function to test for a 'cancellation law'

The problem above makes use of three dimensional arrays to structure the data and then works with structures of rank 2 or less. Many of our matrix problems are oriented to the solution of linear equations or some other mathematical use of matrices; structural uses of arrays of rank two or higher open an number of possibilities for programming problems which will be of interest to even the first semester freshman. Board games in general, and in particular chess, provide fertile ground for sowing seeds of array problems. Ask students to generate a chess board and they probably will not produce the *BOARD* function of Figure 3, which is a modification of the function given by Jones(1969). Students usually submit solutions which either use successive catenation of matrices to form the board or which forms a board with all rows the same and then rotates blocks of rows (or columns) to form the board. when given this programming task.

The next assignment was to use the board and to place the pieces on it using the Forsthye notation, so that *POSITION* 'RNBQKBNRPPPPPPPP8888pppppppprnbqkbnr' *GIVES THE BOARD AT THE* beginning of the game as viewed by white. This function is also shown in Figure 3 and uses *BOARD* in its generation of the position display.

```
      ∇ Z←BOARD
[1]     Z←' /⌹'[1+2,(⍉2,((40⍴5<⍳10)∘.≠24⍴3<⍳6),2),2]
      ∇


      ∇ Z←POSITION X;W;V
[1]     W←' 12345678'⍳V\(V←X∊'12345678')/X
[2]     W←,((~V)[(,W∘.≥⍳8)/,(⍳⍴W)∘.+8⍴0]/¯2+5×⍳64)∘.+ ¯1 0 1
[3]     Q←,(Z←BOARD)[3×⍳8;1+⍳40]
[4]     Q[W]←,' ',((~V)/X),[1.5] ' '
[5]     Z[3×⍳8;1+⍳40]← 8 40 ⍴Q
      ∇
```

Figure 3
Function to Generate a Chess Board
and a Position in a Game

There are alternate forms that one could require in specifying the board position to *POSITION*. For example, we could break each row by the separator,/, and denote empty rows by,//, a row with no piece on it, rather than by8. Further, the problem can be related to text editing problems or other contexts of translation on vectors of characters.

Taking the next step we move to the eight queens problem where we seek all possible board arrangements wherein eight queens are positioned such that all squares are covered and no queen may take another. while this problem is discussed in a number of books on programming applications, one of which is Gruenberger and Jaffray(1965), but the use of *APL* affords the study of a number of interesting aspects.

One way to represent a board with the 8 queens on it is as an 8 by 8 matrix of 0's and 1's; another way is to note that the moves available to the queen and the requirement that no queen may take another permits the information to be stored as a vector which is a permutation of ⍳8, with the *I*th value giving the column position for the queen in that row.

A crude way to proceed is to generate 8?8 and test if it is a solution. When a student begins this way he is soon led to look for a more orderly method of attacking the problem. Figure 4 gives the function *QUEENS* which in turn calls the recursive function *GENERATE* producing a matrix result such that each row is a solution.

```
  ∇ ANS←QUEENS;O;COL;DOWN;UP;X;K;N
    ⍝USES THE ORIGIN FUNCTION
    O←ORIGIN 0
    COL←X←8⍴1
    DOWN←UP←15⍴1
    K←N←0
    ANS←⍳0
    GENERATE
    O←ORIGIN O
  ∇
```

```
  ∇ Z←L IF R
    Z←R/L
  ∇
```

```
  ∇ GENERATE;H
     ⍝ USES IF
     H←0
   REPEAT:→INC IF~COL[H]∧UP[7+N-H]∧DOWN[N+H]
    SETX[N]←H
    COL[H]←UP[7+N-H]←DOWN[N+H]←0
    N←N+1
    →SOLN IF N=8
    GENERATE
   REMOVE:N←N-1
    COL[H]←UP[7+N-H]←DOWN[N+H]←1
   INC:→REPEAT×8≠H←H+1
   SOLN:ANS←ANS,[0] X
    →REMOVE
  ∇
```

Figure 4
The Eight Queens Problem

The result from *QUEENS* has 92 rows and any row may be printed as a board position by '⎕⌹'[*X*∘.=⍳8] where *X* is any row from the result. The follow on problem deals with the fact that because of symmetries, due to rotations of the board by ninety degrees or veiwing the board from the back, the 92 solutions can be reduced to 12 families of solutions. The class can now be given the problem of taking the 92 solutions and finding the families, studying the relationships between the rotations and reversals of the matrix form of the solution and the the corresponding operations on the vector form of the representation. these can be used as supplemental tests to cut down the work in generating the board positions. We leave this as an exercise to the interested reader.

4. SUMMARY

A greater emphasis on arrays and applications which are rich in data structuring is one aspect of what we have been advocating above. Not only is it possible capture the student's interest with problems which are simple and non numerical in nature, it is possible to return to the same problem for more challenging mathematical concepts to provide more challenging programming tasks. The other side of the same argument favors iteration in the programming experience which can be used to explore variations on a coding theme to offer new insight into formulation of the problem. It is conjectured that this leads to better *APL* programming where better is not necessarily one-liner solutions, but rather programs which use arrays to the greatest extent possible.

REFERENCES

Falkoff, A.D. and Iverson, K.E.(1968) *APL*\360 Users Manual, IBM Corp.,T.J.Watson Research Center, Yorktown Heights,NY form GH 20-0683, replaced by *APL*\360 OS and DOS User's Manual, IBM Corp., form SH 20-0906-0 first ed. December 1970

Gilman,L.C. and Rose, A.J. (1970) *APL*\360An Interactive Approach, J.Wiley, New York ,NY

Grey, L.D., A Course in*APL*\360with Applications, Addison-Wesley, Reading, MA

Gruenberger,F. and Jaffray,G(1965) Problems for Computer Solution J.Wiley, New York,NY jones, w.j.(1969) *CHESSBD* Program, *APL* Quote-Quad, 1,4(July)p.13

Katzan, H,Jr. (1970) *APL*Programming and Computing Techniques, Van Nostrand-Reinhold, New York ,NY

Katzan,H.,Jr.(1971) *APL*User's Guide, Van Nostrand-Reinhold New York, NY

Kolsky,H.G.(1969)Problem Formulation Using *APL*,IBM Systems Journal 8,3,p.204

pakin,s.(1968-1st Ed.,1972-2nd Ed.) *APL*\360Reference Manual,Science Research Associates, Palo Alto, CA

Prager,W. (1971) An Introduction to *APL*, Allyn and Bacon, Boston, MA

Sykes,R.A.,Jr.(1973) The Use and MIsuse of *APL*, Effective Coding Techniques, SHARE XL, March 7, Denver, CO

The Variety of Alternative Definitions of a Simple Function

E. E. McDonnell
IBM Scientific Center
3401 Market Street
Philadelphia, PA 19063

1. The problem

The comments made in this paper derive from a study of a variety of solutions to a small problem in APL.

The problem is to create the first *M* rows and *N* columns of the table formed by arranging the integers diagonally:

```
   1  2  4  7 11
   3  5  8 12
   6  9 13
  10 14
  15
```

and so forth. The integers can begin with either one or zero, and may go either up or down the diagonals, since any of these four patterns is easily changed to another by transposition or addition.

In any rectangular portion of the table, the entries are no longer continuous, since the rectangular constraint cuts off some of the elements in the continuous series. For example, when *M*←3 and *N*←5, the desired table is:

```
 1  2  4  7 11
 3  5  8 12 17
 6  9 13 18 24.
```

Some of the many solutions which have come to my attention are included in the figures, each solution evaluated for the case *M*←3, *N*←5. The rest of this paper consists of comments on the algorithmic devices used in the solutions.

2. Algebraic solutions

The solutions grouped under the heading "algebraic" are shown in Figures 1 through 11. Their common character is that they solve the problem by finding an expression for the typical term of the result. They differ in the expressions they use.

2.1 Factoring inside outer products

Two almost identical algorithms may be widely separated in efficiency. One common cause of such wide separation is shown in Figures 2 and 3. In each, the problem is solved by adding two matrices formed with the outer product sum of two vectors and a scalar addition. The difference in the two is the order in which the sums are taken. In the first, the outer product sum is produced first, and then the scalar addition occurs. In the second, the scalar addition occurs before the outer product is formed. The associativity of addition assures us that the results are the same in the two cases. If A and B each have 100 elements, the scalar addition takes place 10,000 times in the first case, but only 100 times in the second.

2.2 Triangular numbers

Many of the algebraic solutions build the result by using a matrix of triangular numbers, arranged in skew fashion:

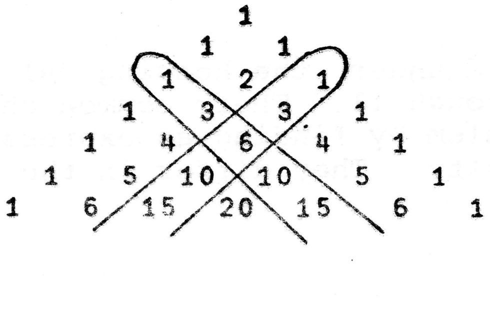
$$\begin{matrix} 0 & 1 & 3 & 6 & 10 \\ 1 & 3 & 6 & 10 & 15 \\ 3 & 6 & 10 & 15 & 21 \end{matrix}$$

The triangular numbers can be generated in a great variety of ways. The N-th triangular number is defined as the sum of the integers to N. The first several triangular numbers are 1 3 6 10 15, formed from 1, 1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5. The formula for the sum of such a series is `0.5×N×(1+N)`. An equivalent formulation could be `0.5×N+N*2`, obtained by performing the multiplication `N×(1+N)`, but since this substitutes exponentiation for multiplication, it is not an attractive alternative.

Since the expression is a quadratic polynomial, Horner's method can be used in its evaluation. In this form, we have `N×0.5+N×0.5`.

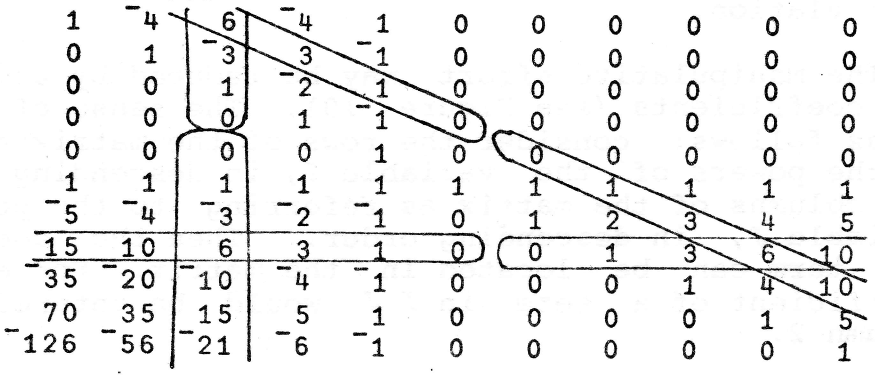
Again because it is a polynomial, an expression such as **`N⊥ 0.5 0.5 0`** can be used.

The triangular numbers appear in the Pascal triangle:



This suggests the use of the binomial coefficient function ! to generate the numbers. The diagonal going down from right to left may be constructed by (*N*-1)!(*N*+1), and the diagonal going down from left to right by 2!(*N*+1). The second form is simpler, since it consists of two dyadic function evaluations, compared to the three of the first expression.

There is more richness in the binomial function than is indicated by the Pascal triangle. Forming a table using an integer vector containing negative as well as positive integers, and the binomial function, we find the triangular number pattern four more times, twice directly and twice alternating:



```
        ⎕←W←¯6+⍳11
¯5 ¯4 ¯3 ¯2 ¯1 0 1 2 3 4 5
        W∘.!W
   1  ¯4   6  ¯4   1   0   0   0   0   0   0
   0   1  ¯3   3  ¯1   0   0   0   0   0   0
   0   0   1  ¯2   1   0   0   0   0   0   0
   0   0   0   1  ¯1   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0   0   0
   1   1   1   1   1   1   1   1   1   1   1
  ¯5  ¯4  ¯3  ¯2  ¯1   0   1   2   3   4   5
  15  10   6   3   1   0   0   1   3   6  10
 ¯35 ¯20 ¯10  ¯4  ¯1   0   0   0   1   4  10
  70  35  15   5   1   0   0   0   0   1   5
¯126 ¯56 ¯21  ¯6  ¯1   0   0   0   0   0   1
```

The binomial function thus provides six formulations for the triangular numbers:

```
2!N+1
(N-1)!N+1
|(N-1)!¯3
(-N+2)!-N
|(-N+2)!¯3
2!-N
```

The last is the most efficient, since only a monadic function and a dyadic function with a constant left argument have to be evaluated. Figures 4 through 7 show uses of the binomial function in solving the problem.

2.3 Bivariate polynomial evaluation

The solution to the problem may be expressed algebraically in terms of the row and column coordinates. This gives rise to a polynomial in two variables, and we are faced with the problem of how to evaluate such a polynomial.

An expression for the element in row I, column J of the result is given by:

```
.5×(I*2)+(2×I×J)+(J*2)+(3×I)+J                    (1)
```

In terms of the arguments to the function that we want, this could be rewritten: (see Figure 8)

```
0.5×((⍳M)∘.+⍳N)*2)+(3×⍳M)∘.+⍳N
```

The two outer products in this formulation are each for the full size of the result.

By using Horner's method, a reduction in effort is made possible (see Figure 9). In this, only one outer product is required, but there is a fairly large amount of manipulation.

The manipulative effort may be reduced by using a matrix of coefficients (see Figure 10). The sense of this matrix is as follows: consider the rows of the matrix as referring to the powers of the variable *I*, in descending order, and the columns of the matrix as referring to the powers of the variable *J*, in descending order. Then the coefficient of any term can be located in the matrix. For example, the coefficient of a term in *I*×*J* would be entered in row 2, column 2.

```
     J*2 J*1 J*0
I*2
I*1
I*0
```

The elements of the matrix may be determined by inspection of expression (1) above:

```
0   0   0.5
0   1   0.5
0.5 1.5 0
```

Figure 10, which makes use of this, requires two outer products, but the size of the outer product is determined by the degree of the polynomial. In the case at hand, the right argument in each outer product is of length three.

The algorithm of Figure 10, after the use of the two fairly simple outer products, requires two inner products and a transpose to accomplish its result.

The base value function may be substituted for the matrix product and the formation of the powers of the variables.

To do this, the variables have to be reshaped to have a last dimension of 1. With this done, the function can be expressed compactly, and with only trivial manipulation (see Figure 11).

More generally, evaluating a multivariate polynomial requires a coefficient array with as many axes as there are variables, and each axis having a length as required by the degree with which the associated variable enters into the polynomial. For example, if variables X, Y, and Z enter into a polynomial with degrees 3, 5, and 2, respectively, the coefficient matrix required would have size 4 6 3. It is necessary to transpose the result of each use of the base value function except the last, in order to prepare the partial result for the next use of base value.

3. Structural solutions

Although all the solutions combine uses of algebra and some sense of the structure of the problem, the techniques described in Section 2 are derived from an algebraic understanding of the problem, while the techniques described in this section are based on a structural understanding of the problem.

The programming elements which predominate in Figures 1 through 11 are multiplication, matrix product, exponentiation, and outer product. In this section the programming elements most heavily used are take, catenate, and scan. A study of Figures 12 through 17 suggests that the solver determined what form was wanted, decomposed the form into pieces that were easily computed, united these by catenation, and then with a scan or a take produced the desired result.

3.1 Scan

Many of the structural solutions depend on the use of sum-scan to attain their end. Since the triangular numbers may be defined in terms of sum-reduction, this is to be expected. What may be unexpected is the great variety of ways in which sum-scan can be used in arriving at a solution (Figures 12 through 15).

3.2 Rho-rotate

The solutions of Figures 16 and 17 demonstrate a property of dyadic rho which may be used to advantage when a rotation of the rows of a matrix is done in a regular progression.

Notice that the argument to rotate in Figure 16 is 0 1 2. This makes it possible to achieve the effect of rotation by the use of dyadic rho in Figure 17.

The same strategy underlies the device used to obtain an identity matrix of order N:

```
(N,N)ρ(N+1)↑1
```

The solution shown in Figure 17, like that of Figure 16, can be inefficient in its use of space. The solutions of Figures 1 through 15 generate only as many elements as are required to form the result, that is, *M×N*. In terms of space, these solutions can handle a 1250 by 2 case as easily as a 50 by 50 case: 2500 elements are generated in both; but the solutions of Figures 16 and 17 would generate 50×99 or 4900 elements for the 50 by 50 case, and 2250×2252, or 1,563,570 elements in the 1250 by 2 case.

Acknowledgments: Solutions by the following were used in preparing this paper:

P. S. Abrams
L. M. Breed
G. C. Driscoll
A. D. Falkoff
M. Halpern
K. E. Iverson
H. J. Smith, Jr.
K. L. Solheim
R. Sykes
S. Tsao

P. C. Berry, A. D. Falkoff, and K. E. Iverson reviewed various drafts of this paper, and made many helpful suggestions.

Figure 1

```
R←(A∘.×B)+(0.5×A×3+A←⍳M)∘.+0.5×B×1+B←⍳N

   (A∘.×B)+(0.5×A×3+A←⍳3)∘.+0.5×B×1+B←⍳5

   (A∘.×B)+(0.5×0 1 2×3+0 1 2)∘.+0.5×0 1 2 3 4×1+0 1 2 3 4

   (A∘.×B)+(0.5× 0 1 2 × 3 4 5)∘.+0.5× 0 1 2 3 4 × 1 2 3 4 5

   (A∘.×B)+(0.5× 0 4 10)∘.+0.5× 0 2 6 12 20

   (0 1 2 ∘.× 0 1 2 3 4)+ 0 2 5 ∘.+ 0 1 3 6 10

   0 0 0 0 0    0  1  3  6 10
   0 1 2 3 4 +  2  3  5  8 12
   0 2 4 6 8    5  6  8 11 15

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 2

```
R←0.5×(¯1+A∘.-B)+(1+(A←⍳M)∘.+B←⍳N)*2

  0.5×(¯1+A∘.-B)+(1+(A←⍳3)∘.+B←⍳5)*2

  0.5×(¯1+0 1 2∘.-0 1 2 3 4)+(1+0 1 2∘.+0 1 2 3 4)*2

             0 ¯1 ¯2 ¯3 ¯4       0 1 2 3 4
  0.5×(¯1+   1  0 ¯1 ¯2 ¯3) +(1+ 1 2 3 4 5)*2
             2  1  0 ¯1 ¯2       2 3 4 5 6

        ¯1 ¯2 ¯3 ¯4 ¯5   1 2 3 4 5
  0.5×   0 ¯1 ¯2 ¯3 ¯4 +(2 3 4 5 6)*2
         1  0 ¯1 ¯2 ¯3   3 4 5 6 7

        ¯1 ¯2 ¯3 ¯4 ¯5   1  4  9 16 25
  0.5×   0 ¯1 ¯2 ¯3 ¯4 + 4  9 16 25 36
         1  0 ¯1 ¯2 ¯3   9 16 25 36 49

         0  2  6 12 20
  0.5×   4  8 14 22 32
        10 16 24 34 46

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 3

```
R←0.5×(A∘.-1+B)+((A←⍳M)∘.+1+B←⍳N)*2

  0.5×(A∘.-1+B)+((A←⍳3)∘.+1+B←⍳5)*2

  0.5×(A∘.-1+0 1 2 3 4)+(A∘.+1+0 1 2 3 4)*2

  0.5×(0 1 2∘.-1 2 3 4 5)+(0 1 2∘.+1 2 3 4 5)*2

        ¯1 ¯2 ¯3 ¯4 ¯5     1 2 3 4 5
  0.5×   0 ¯1 ¯2 ¯3 ¯4 + (2 3 4 5 6)*2
         1  0 ¯1 ¯2 ¯3     3 4 5 6 7

        ¯1 ¯2 ¯3 ¯4 ¯5   1  4  9 16 25
  0.5×   0 ¯1 ¯2 ¯3 ¯4 + 4  9 16 25 36
         1  0 ¯1 ¯2 ¯3   9 16 25 36 49

         0  2  6 12 20
  0.5×   4  8 14 22 32
        10 16 24 34 46

0 1  3  6 10
2 4  7 11 16
5 8 12 17 23
```

Figure 4

```
R←(!⌿ 0 2 ∘.+(⍳M)∘.+⍳N)-(M,N)⍴⍳N

   (!⌿ 0 2 ∘.+(⍳3)∘.+⍳5)- 3 5 ⍴⍳5

   (!⌿ 0 2 ∘.+ 0 1 2 ∘.+ 0 1 2 3 4)- 3 5 ⍴ 0 1 2 3 4

                   0 1 2 3 4                 0 1 2 3 4
   (!⌿ 0 2 ∘.+ 1 2 3 4 5 )               - 0 1 2 3 4
                   2 3 4 5 6                 0 1 2 3 4

       0 1 2 3 4
       1 2 3 4 5
       2 3 4 5 6                           0 1 2 3 4
   (!⌿               )           -         0 1 2 3 4
       2 3 4 5 6                           0 1 2 3 4
       3 4 5 6 7
       4 5 6 7 8

    1  3  6 10 15                      0 1 2 3 4
    3  6 10 15 21           -          0 1 2 3 4
    6 10 15 21 28                      0 1 2 3 4

 1  2  4  7 11
 3  5  8 12 17
 6  9 13 18 24
```

Figure 5

```
R←(2!-(1+⍳M)∘.+⍳N)-(M,N)⍴⍳N

  (2!-(1+⍳3)∘.+⍳5)- 3 5 ⍴⍳5

  (2!-(1+ 0 1 2)∘.+ 0 1 2 3 4)- 3 5 ⍴⍳5

        1 2 3 4 5
  (2!-2 3 4 5 6)   -   3 5 ⍴⍳5
        3 4 5 6 7

   1  3  6 10 15       0 1 2 3 4
   3  6 10 15 21   -   0 1 2 3 4
   6 10 15 21 28       0 1 2 3 4

1  2  4  7 11
3  5  8 12 17
6  9 13 18 24
```

Figure 6

```
R←(2!-A∘.+B)+(A←⍳M)∘.+0×B←⍳N

  (2!-(⍳3)∘.+⍳5)+(⍳3)∘.+0×⍳5

  (2!- 0 1 2 ∘.+ 0 1 2 3 4) + 0 1 2 ∘.+ 0 0 0 0 0

       0 1 2 3 4                0 0 0 0 0
  (2!- 1 2 3 4 5  )       +     1 1 1 1 1
       2 3 4 5 6                2 2 2 2 2

  0  1  3  6 10                 0 0 0 0 0
  1  3  6 10 15           +     1 1 1 1 1
  3  6 10 15 21                 2 2 2 2 2

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 7

```
R←(2!-C)+⌊\C←(⍳M)∘.+⍳N

  (2!-C)+⌊\C← 0 1 2 ∘.+ 0 1 2 3 4

              0 1 2 3 4
  (2!-C)+⌊\C← 1 2 3 4 5
              2 3 4 5 6

      0 1 2 3 4      0 0 0 0 0
  (2!-1 2 3 4 5 ) + 1 1 1 1 1
      2 3 4 5 6      2 2 2 2 2

   0  1  3  6 10    0 0 0 0 0
   1  3  6 10 15 + 1 1 1 1 1
   3  6 10 15 21    2 2 2 2 2

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 8

```
R←0.5×((A∘.+B)*2)+(3×A←⍳M)∘.+B←⍳N

  0.5×((⍳3)∘.+⍳5)*2)+(3× 0 1 2 )∘.+ 0 1 2 3 4

                              0 3 6

      (0 1 2 ∘.+ 0 1 2 3 4)*2

       0 1 2 3 4                  0  1  2  3  4
  0.5×(1 2 3 4 5)*2   +           3  4  5  6  7
       2 3 4 5 6                  6  7  8  9 10

      0  1  4  9 16              0  1  2  3  4
  0.5×1  4  9 16 25     +        3  4  5  6  7
      4  9 16 25 36              6  7  8  9 10

       0  2  6 12 20
  0.5× 4  8 14 22 32
      10 16 24 34 46

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 9

```
R←(⍉(N,M)ρA×1.5+0.5×A)+((M,N)ρB)×(A←ιM)∘.+0.5+0.5×B←ιN

  (⍉5 3ρA×1.5+0.5×A)+(3 5ρB)×(A←ι3)∘.+0.5+0.5×B←ι5

  (⍉5 3ρρ(ι3)×1.5+0.5×ι3)+(3 5ρι5)×(ι3)∘.+0.5×0 1 2 3 4

  (⍉5 3ρ(ι3)×1.5+0.5×ι3)+(3 5ρι5)×0 1 2∘.+0 0.5 1 1.5 2

                              0 1 2 3 4   0.5 1   1.5 2   2.5
  (⍉5 3ρ(ι3)×1.5+0.5×0 1 2) + 0 1 2 3 4 × 1.5 2   2.5 3   3.5
                              0 1 2 3 4   2.5 3   3.5 4   4.5

                             0  1  3  6 10
  (⍉5 3ρ(ι3)×1.5+ 0 0.5 1) + 0  2  5  9 14
                             0  3  7 12 18

                             0  1  3  6 10
  (⍉5 3ρ(ι3)× 1.5 2 2.5)   + 0  2  5  9 14
                             0  3  7 12 18

                              0  1  3  6 10
  (⍉5 3ρ 0 1 2 × 1.5 2 2.5) + 0  2  5  9 14
                              0  3  7 12 18

                         0  1  3  6 10
  (⍉5 3ρ 0 2 5)       +  0  2  5  9 14
                         0  3  7 12 18

    0 2 5
    0 2 5     0  1  3  6 10
  (⍉0 2 5)  + 0  2  5  9 14
    0 2 5     0  3  7 12 18
    0 2 5

  0 0 0 0 0   0  1  3  6 10
  2 2 2 2 2 + 0  2  5  9 14
  5 5 5 5 5   0  3  7 12 18

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 10

```
R←((⍳M)∘.* 2 1 0)+.×⍉((⍳N)∘.* 2 1 0)+.×U

   ((⍳3)∘.* 2 1 0)+.×⍉((⍳5)∘.* 2 1 0)+.×U

   (0 1 2 ∘.* 2 1 0)+.×⍉(0 1 2 3 4 ∘.* 2 1 0)+.×U

                    0  0  1
   0 0 1            1  1  1        0    0    0.5
   1 1 1   +.×⍉     4  2  1  +.×   0    1    1.5
   4 2 1            9  3  1        0.5  1.5  0
                   16  4  1

                  0.5  1.5   0
   0 0 1          0.5  2.5   1
   1 1 1  +.×⍉    0.5  3.5   3
   4 2 1          0.5  4.5   6
                  0.5  5.5  10

   0 0 1       0.5  0.5  0.5  0.5  0.5
   1 1 1  +.×  1.5  2.5  3.5  4.5  5.5
   4 2 1       0    1    3    6   10

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 11

```
R←((M,1)⍴⍳M)⊥⍉((N,1)⍴⍳N)⊥U

  (3 1⍴⍳3)⊥⍉(5 1⍴⍳5)⊥U

            0
  0         1   0      0     0.5
  1  ⊥⍉     2 ⊥ 0      1     0.5
  2         3   0.5  1.5     0
            4

            0.5   1.5   0
  0         0.5   2.5   1
  1  ⊥⍉     0.5   3.5   3
  2         0.5   4.5   6
            0.5   5.5  10

  0    0.5   0.5   0.5   0.5   0.5
  1 ⊥  1.5   2.5   3.5   4.5   5.5
  2    0     1     3     6    10

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 12

```
R←+\(A++\A),1+(A←⍳M)∘.+⍳N-1

  +\((⍳3)++\⍳3),1+(⍳3)∘.+⍳5-1

  +\(0 1 2 ++\ 0 1 2),1+ 0 1 2 ∘.+ 0 1 2 3

                       0 1 2 3
  +\(0 1 2 + 0 1 3),1+ 1 2 3 4
                       2 3 4 5

             1 2 3 4
  +\ 0 2 5 , 2 3 4 5
             3 4 5 6

     0 1 2 3 4
  +\ 2 2 3 4 5
     5 3 4 5 6

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 13

```
R←+\(A++\A),(1+A←⍳M)∘.+⍳N-1

   +\((⍳3)++\⍳3),(1+⍳3)∘.+⍳5-1

   +\(0 1 2 ++\ 0 1 2), 1 2 3 ∘.+ 0 1 2 3

                       1 2 3 4
   +\(0 1 2 + 0 1 3), 2 3 4 5
                       3 4 5 6

               1 2 3 4
   +\ 0 2 5 , 2 3 4 5
               3 4 5 6

       0 1 2 3 4
   +\ 2 2 3 4 5
       5 3 4 5 6

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 14

```
R←+\(+\⍳M),(¯1+⍳M)∘.+⍳N-1

  +\(+\⍳3),(¯1+⍳3)∘.+⍳5-1

  +\(+\ 1 2 3),(¯1+ 1 2 3)∘.+⍳4

  +\(1 3 6), 0 1 2 ∘.+ 1 2 3 4

                   1 2 3 4
  +\ 1 3 6 ,       2 3 4 5
                   3 4 5 6

       1 1 2 3 4
  +\   3 2 3 4 5
       6 3 4 5 6

1  2  4  7 11
3  5  8 12 17
6  9 13 18 24
```

Figure 15

```
R←+⍀(+\⍳N),[0](1+⍳M-1)∘.+1+⍳N

  +⍀(+\⍳5),[0](1+⍳3-1)∘.+1+⍳5

  +⍀(+\0 1 2 3 4),[0](1+⍳2)∘.+1+0 1 2 3 4

  +⍀ 0 1 3 6 10,[0](1+ 0 1)∘.+ 1 2 3 4 5

  +⍀ 0 1 3 6 10 ,[0] 1 2 ∘.+ 1 2 3 4 5

                      2 3 4 5 6
  +⍀ 0 1 3 6 10 ,[0] 3 4 5 6 7

     0  1  3  6 10
  +⍀ 2  3  4  5  6
     3  4  5  6  7

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 16

```
R←(M,N)↑A⌽(A←⍳M)∘.++\⍳M+N-1

   3 5 ↑ 0 1 2 ⌽ 0 1 2 ∘.++\⍳3+5-1

   3 5 ↑ 0 1 2 ⌽ 0 1 2 ∘.++\⍳7

   3 5 ↑ 0 1 2 ⌽ 0 1 2 ∘.++\ 0 1 2 3 4 5 6

   3 5 ↑ 0 1 2 ⌽ 0 1 2 ∘.+ 0 1 3 6 10 15 21

                       0  1  3  6 10 15 21
   3 5 ↑ 0 1 2 ⌽ 1  2  4  7 11 16 22
                       2  3  5  8 12 17 23

           0  1  3  6 10 15 21
   3 5 ↑ 2  4  7 11 16 22  1
           5  8 12 17 23  2  3

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

Figure 17

```
R←(M,N)↑(M,M+N)⍴(⍳M)∘.++\⍳M+N-1

  3 5 ↑(3,3+5)⍴ 0 1 2 ∘.++\⍳3+5-1

  3 5 ↑ 3 8 ⍴ 0 1 2 ∘.++\⍳7

  3 5 ↑ 3 8 ⍴ 0 1 2 ∘.++\ 0 1 2 3 4 5 6

  3 5 ↑ 3 8 ⍴ 0 1 2 ∘.+ 0 1 3 6 10 15 21

                 0  1  3  6 10 15 21
  3 5 ↑ 3 8 ⍴ 1  2  4  7 11 16 22
                 2  3  5  8 12 17 23

            0  1  3  6 10 15 21  1
  3 5 ↑ 2  4  7 11 16 22  2  3
            5  8 12 17 23  0  1  3

0  1  3  6 10
2  4  7 11 16
5  8 12 17 23
```

COMPUTERS IN INSTRUCTION
AT THE
COAST COMMUNITY COLLEGES

BY

JOHN CLARK and DICK MERCER

The attached terminal printouts represent simulated Student Terminal Sessions. They provide examples of the variety of CAI segments used in the Coast Community College District. Each Segment presupposes some advance preparation on the part of the student. Most of the printouts do not include the initial interaction in which the student may obtain specific instruction on the control of the segment. In each segment the student input is underlined.

MEDTERMS

A drill-and-practice segment dealing with medical terminology and the meanings of commonly used medical prefixes. Questions and choices may be presented randomly. A retort is given for each student choice. A series of 20 similar segments have been developed which use a computer-driven Microfiche Projector.

SPANISH

A comprehensive series of instructional segments covering the whole of Spanish grammar, including the formation of all verb tenses; gender, plural and uses of nouns, pronouns, adjectives and articles; uses of adverbs, prepositions, and verbs; language syntax and other items related to Spanish sentence strategies. The series includes a 3,000 word vocabulary and approximately 200 idioms. The organizational format follows that of "Foundation Course in Spanish" by Turk and Espinosa. There are 193 segments providing 100 hours of instruction. Eleven hundred (1,100) student hours resulted from 1,200 student contacts during the Fall semester.

MODIFIERS

A tutorial program in English which requires the student to find and reposition a misplaced modifier in a sentence. There are 30 segments providing 21 hours of English instruction in such areas as spelling, paragraph coherence, sentence fragments, modifiers, transitions and often confused words. Eleven hundred (1,100) student hours resulted from 1,300 student contacts during the Fall semester.

PROBLEM
One of a series of segments which present word problems, and allows the student several options in its solution. The student may supply the solution, request more data, or enter an analysis mode in which he is asked to perform in a manner similar to what he would in an office session with an instructor. For this example the instructions, a problem and a series of analysis steps are shown. The software for this segment is suitable for use with Physics, Chemistry, or other subjects using word problems.

LIMIT
A segment designed to help a Calculus student with delta-epsilon limit proofs for polynomial functions. This is the first of a series of segments devoted to limits and limit proofs. The student has the option in this segment of requesting the proof and an explanation of any step in the proof. The student inputs the coefficients of the polynomial of his choice and the point where the limit is to be evaluated.

SUPERGRAPH
One of a series of plotting routines designed to aid the student in the discovery of mathematical relationships. The student may choose the function to be plotted and the domain of evaluation. Included in this workspace are routines to provide polar and parametric plots as well as routines to locate the real zeros of the function.

ZPLOT
A segment which visually displays complex roots and multiple roots. The student may use standard textbook notation for defining the function. By observing the function as the junction of various regions, the zeros may be obtained. In this example, a multiple root occurs at 4, real roots occur at -3.1 and -0.8, and complex roots of (1±1.5i) and (4.7±3.5i). Other segments in this workspace will locate the zeros to a higher degree of accuracy.

LEASTFIT
A physics workspace used for analysis of experimental data. In this example the student receives both descriptive statistics and a histogram comparing his data with a theoretical Gaussian Distribution having the same mean and standard deviation. The * printed in this Histogram represents points through which the normal curve would pass. Included in this workspace are other curve-fitting and analysis routines.

ELECTRONICS
A simulation segment which provides the student an opportunity to trouble-shoot a randomly generated failure in a given circuit. The student is given the symptoms of the failure and has the opportunity to test and/or replace various components at his descresion. The trouble persists until the failing component is replaced.

```
4.  WHICH PREFIX MEANS 'HEART'?

          CEPHAL-
          CONTRA-
          CARDIA-
          CHOL-

▯:  CARDIA

CORRECT-THAT WAS CLOSE!  AS USED IN CARDIOGRAM WHICH IS A TRACING OF HEART
RHYTHM AND PATTERN.  PROCEED TO NEXT ITEM.


5.  WHICH OF THE FOLLOWING PREFIXES DENOTES A 'BLOOD VESSEL'?

          CEPHAL-
          CHOLE-
          BRADY-
          ANA-
          ANGIO-

▯:  ANA

NO, NO!  ANA- MEANS TO 'RENEW OR UNITE' AS IN ANASTOMOSIS WHICH MEANS A JOINING
OF TWO VESSELS OR NERVES.  TRY AGAIN.

▯:  BRADY

INCORRECT!  BRADY- MEANS 'SLOW' AS IN BRADYCARDIA WHICH MEANS SLOW HEART.
TRY AGAIN.

▯:  ANA
YOU HAVE MADE THIS CHOICE BEFORE
▯:  ANGIO

CORRECT!  AS IN ANGIOSPASM, WHICH MEANS A SPASMODIC CONTRACTION OF BLOOD
VESSELS.  PLEASE CONTINUE.


6.  WHICH PREFIX MEANS 'AGAINST'?

          BRADY-
          CON-
          BI-
          ANTI-
          CLEIDO-

▯:  CONTRA
YOU MUST CHOOSE ONE OF THE ABOVE CHOICES
▯:  ANTI

HOW NICE!  AS IN ANTIBIOTIC, WHICH IS A SUBSTANCE AGAINST VACTERIAL LIFE.
PLEASE CONTINUE.


7.  WHICH PREFIX MEANS 'CARTILAGE'?

          CEPHAL-
          OSTEO-
          CARDIA-
          CHOL-
          CHONDRA-
```

```
PLEASE NOTE :

      °........THE SMALL CIRCLE IS USED TO INDICATE INCORRECT SPELLING.
                EX.: TRIAL MISPELLED = TRI°L.

      ↓........THE ARROW POINTS TO INCORRECT PUNCTUATION.
                EX.: ME LLAMO PEPE↓  (NO PERIOD).

      _........UNDERLINE LETTERS THAT NEED AN ACCENT OR TILDE.
                EX.: LAPIZ  MANANA

WHICH SECTION DO YOU WANT?...  5

GIVE IN SPANISH.  DON'T USE THE SUBJECT PRONOUNS.

WE SPEND A SHORT TIME HERE EVERY DAY.
?...  GASTAMOS UN ????? AQUI TODOS LOS DIAS.
      °AS°°°° UN °°°° AQU° TODOS LOS D°AS.

WE SPEND A SHORT TIME HERE EVERY DAY.
?...  PASAMOS UN RATO AQUI TODOS LOS DIAS.
!ESO ES!

WHAT TIME IS IT?
?...  QUE HORA ES?
      ↓QUE HORA ES?

WHAT TIME IS IT?
?...  ?QUE HORA ES?
THAT'S CORRECT!

I HAVE A LOT OF TIME TO STUDY.
?...  HELP

DICK, HOW ABOUT:

         TENGO MUCHO TIEMPO PARA ESTUDIAR.


I'M NOT GOING WITH MY MAMA THIS TIME.
?...  NO     VOY CON       MI MAMA ESTA VEZ.
!BRAVISIMO!

I GO TWO TIMES EVERY WEEK.
?...  VOY DOS TIEMPOS CADA SEMANA.
      VOY DOS °°°°° CADA SEMANA.

I GO TWO TIMES EVERY WEEK.
?...  VOY DOS VECES CADA SEMANA.
!ESO ES!

DO YOU ALWAYS ARRIVE ON TIME?
?...  STOP

OK DICK. WE'LL STOP HERE FOR NOW.

DO YOU WANT TO SEE THE SCORE BOARD?...SI

LESSON                5
TIME                  4
PROBLEMS TRIED        5
ANSWERED CORRECTLY    4
PER CENTAGE          80
```

```
TOMORROW   MANANA

MANY   MUY
MANY   MUCHO
**ANSWER:  MANY-MUCHOS

NORTH AMERICAN   AMERICANO
NORTH AMERICAN   NORTEAMERICANO

IN THE AFTERNOON (HOUR GIVEN)
                         DE LA TARDE

MANY   MUCHOS

TIME   LA VEZ
TIME   LA HORA

CHARLES   CARLO
CHARLES   CARLO
**ANSWER:  CHARLES-CARLOS

URUGUAY   EL URUGUAY

THERE IS   HAY

CHARLES   CARLOS

SON   STOP
```

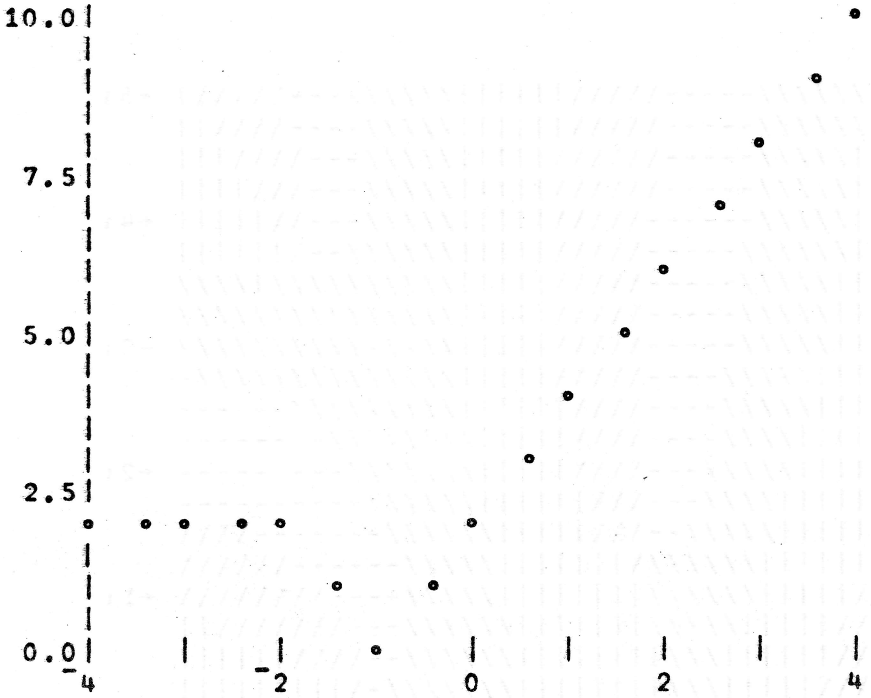
```
BEER SHOULD NOT BE SOLD TO STUDENTS CONTAINING MORE THAN 3.2 PERCENT ALCOHOL.

TYPE THE FIRST AND LAST WORDS OF THE MISPLACED MODIFIER.
▯
SOLD STUDENTS


NO, DICK, YOU HAVEN'T GOT THE RIGHT MODIFIER *****
TRY AGAIN.
▯
CONTAINING ALCOHOL


GOOD, YOU HAVE IDENTIFIED THE MODIFIER.


WHERE SHOULD 'CONTAINING MORE THAN 3.2 PERCENT ALCOHOL'
BE POSITIONED IN THE SENTENCE?
▯
BEER∧



 BEER CONTAINING MORE THAN 3.2 PERCENT ALCOHOL SHOULD NOT BE SOLD TO STUDENTS.
VERY GOOD, DICK, ON TO THE NEXT QUESTION.


SHE WORE A RIBBON IN HER HAIR WHICH WAS A LIGHT PINK.

TYPE THE FIRST AND LAST WORDS OF THE MISPLACED MODIFIER.
▯
WHICH PINK


GOOD, YOU HAVE IDENTIFIED THE MODIFIER.


WHERE SHOULD 'WHICH WAS A LIGHT PINK'
BE POSITIONED IN THE SENTENCE?
▯
SHE∧



 SHE WHICH WAS A LIGHT PINK WORE A RIBBON IN HER HAIR.

NO, DICK, THE PROBLEM WITH THE SENTENCE IS THAT HER HAIR APPEARS TO BE 'LIGHT
PINK' WHEN THIS SHOULD BE A DESCRIPTION OF WHAT?  TRY AGAIN.


WHERE SHOULD 'WHICH WAS A LIGHT PINK'
BE POSITIONED IN THE SENTENCE?
▯
RIBBON∧



 SHE WORE A RIBBON WHICH WAS A LIGHT PINK IN HER HAIR.
VERY GOOD, DICK, ON TO THE NEXT QUESTION.
```

```
        )91051
051) 21.41.49 10/20/72 P510CC

   A P L \ 3 7 0

        )LOAD 1234 PROBLEM
SAVED  21.35.51 10/20/72

        GO

IN THIS WORKSPACE YOU WILL BE PRESENTED A PROBLEM.  YOU MAY CHOOSE TO
ANSWER THE PROBLEM DIRECTLY, OR BE ASKED KEY QUESTIONS ABOUT IT.

THERE ARE FOUR DIFFERENT MODES BY WHICH YOU MAY USE THE TERMINAL
IN SOLVING THE PROBLEM.

SOLUTION MODE    TYPE THE LETTER  S  FOLLOWED BY THE NUMBER THAT YOU
                 BELIEVE TO BE THE SOLUTION OF THE PROBLEM.

HINT MODE        TYPE THE LETTER  H  AND I WILL PROVIDE YOU WITH A HINT
                 ABOUT THE SOLUTION OF THE PROBLEM.  IF YOU TYPE   HS
                 I WILL PROVIDE YOU WITH THE SOLUTION, BUT THIS WILL COST
                 YOU 100 POINTS.

INQUIRY MODE     TYPE THE LETTER  I  FOLLOWED BY ONE OF THE LETTERS
                 BELOW TO OBTAIN THE FOLLOWING INFORMATION:

      D     --   DEFINITION OF TERMS USED IN THE PROBLEM
      DD    --   DIRECT DATA INVOLVED IN THE PROBLEM
      ID    --   INDIRECT DATA INVOLVED IN THE PROBLEM
      E     --   THE EQUATION FOR THE SOLUTION
      C     --   THE CONCEPT INVOLVED IN THE PROBLEM
      U     --   THE UNKNOWN INVOLVED IN THE PROBLEM
      F     --   ANY SPECIAL FORMULAE NEEDED FOR THE SOLUTION

ANALYSIS MODE    TYPE THE LETTER  A  AND I WILL ASK YOU THE QUESTIONS
                 YOU SHOULD ASK YOURSELF AS YOU ATTEMPT TO SOLVE THE
                 PROBLEM

YOU MAY REQUEST THE CALCULATOR MODE BY TYPING THE LETTER  C
THE CALCULATOR MODE ASSISTS YOU IN YOUR CALCULATIONS.  TO EXIT THIS MODE
YOU SHOULD TYPE THE WORD   END

YOU START EACH PROBLEM WITH 100 POINTS.  WHEN YOU ASK FOR ANY MODE OTHER
THAN THE SOLUTION OR CALCULATOR MODE, I MAY SUBTRACT POINTS FOR THE HELP
I GIVE YOU.  IN THE ANALYSIS MODE I WILL NOT SUBTRACT POINTS IF YOU ANSWER
MY QUESTIONS CORRECTLY.

YOUR SOLUTION MUST BE CORRECT IN THE SOLUTION MODE BEFORE YOU ARE ALLOWED
TO ADVANCE TO THE NEXT PROBLEM.

TO STOP THIS PROGRAM,  TYPE THE WORD  STOP

GOOD LUCK ON YOUR PROBLEMS!
```

```
HOW MANY GALLONS OF BRANDY CONTAINING 40 PERCENT ALCOHOL
(80 PROOF) SHOULD BE ADDED TO 100 GALLONS OF WINE CONTAINING
12 PERCENT ALCHOL TO PRODUCE AN INTOXICATING BEVERAGE
WHICH HAS 20 PERCENT ALCOHOL (40 PROOF)?

WHEN THE SYMBOL @ APPEARS YOU MAY ENTER YOUR MODE CHOICE.  I WILL
ALWAYS RETURN TO THE MODE CHOICE STATUS EACH TIME.


@


     A
IF X IS THE NUMBER OF GALLONS OF BRANDY, WRITE AN EXPRESSION INVOLVING
X WHICH IS EQUIVALENT TO THE NUMBER OF GALLONS OF ALCOHOL IN THE BRANDY.

.4X
BULLY GOOD
@


     A
HOW MANY GALLONS OF ALCOHOL ARE IN THE WINE?

12
YOU ARE CORRECT ON THIS PORTION OF THE ANALYSIS.
@


     A
REMEMBERING THAT X IS THE NUMBER OF GALLONS OF BRANDY, WRITE AN EXPRESSION
INVOLVING X WHICH EXPRESSED THE NUMBER OF GALLONS OF THE RESULTING
BEVERAGE.

X+100
VERY GOOD
@


     A
WRITE AN EXPRESSION , INVOLVING X, WHICH EXPRESSED THE AMOUNT OF
ALCOHOL IN THE RESULTING VEVERAGE.

.2×(100 + X)
VERY GOOD
@


     A
WRITE AN EXPRESSION, INVOLVING X, WHICH IS EQUIVALENT TO THE
NUMBER OF GALLONS OF ALCOHOL IN THE BRANDY PLUS THE NUMBER OF
GALLONS OF ALCOHOL IN THE WINE.

8+4+2×.2X
EXCELLENT
@


     A
WE KNOW THE TOTAL NUMBER OF GALLONS OF ALCOHOL IN THE BRANDY
AND IN THE WINE ADDED TOGETHER IS EQUAL TO THE TOTAL NUMBER OF
GALLONS OF ALCOHOL IN THE RESULTING BEVERAGE.  WRITE AN EQUATION,
INVOLVING X, WHICH EXPRESSED THIS FACT.

12 + .4X = .2×(X+100)
YOUR EQUATION HAS THE SAME SOLUTION SET AS MINE.  TAKE A LOOK AT MINE
TO SEE IF YOU HAVE THE SAME POINTS THAT I HAVE.
```

LIMIT

```
THIS FUNCTION IS DESIGNED TO ASSIST YOU IN WRITING 'Δ ε' PROOFS FOR
POLYNOMIALS.  WHEN 'F(X)' APPEARS ENTER THE COEFFICIENTS OF THE
POLYNOMIAL.   WHEN 'X → ' APPEARS ENTER THE POINT THE DOMAIN WHERE
THE LIMIT OF THE FUNCTION IS TO BE EVALUATED.


F(X)=
⎕:
      1 0 ¯2 4 2
X →
⎕:
      2


 LIM  X*4 - 2X*2 + 4X + 2 = 18
X→2

 CHOOSE Δ < ε / 59  WHEN 0 < ε < 59  DO YOU WANT PROOF?   YES

WHEN: ε>0; 0<Δ<1; F(X)=X*4 - 2X*2 + 4X + 2

A=2; L=18 HERE IS HOW TO FIND A Δ AS A FUNCTION OF ε  Δ<Φ(ε)

                                   SUCH THAT

            IF 0<|X - A|<Δ     THEN  |F(X) - L|<ε

1)0 < |X - 2| < Δ ; |X*4 - 2X*2 + 4X + 2 - 18| < ε

2)1 < X < 3         ; |X*4 - 2X*2 + 4X - 16| < ε

3)                   |X*3 + 2X*2 + 2X + 8|×|X - 2| < ε

4)|X - 2|    <    ε / |X*3 + 2X*2 + 2X + 8|

 CHOOSE Δ < ε / 59  WHEN 0 < ε < 59
ENTER 0  OR THE LINE NUMBER YOU WISH CLARIFIED
⎕:
      3
IN STEP 3  WE ARE EXPRESSING

|F(X)-L| OR |X*4 - 2X*2 + 4X - 16|, AS THE PRODUCT OF

|X-A| OR |X - 2| WHICH MUST FACTOR EVENLY INTO |F(X)-L|, AND ITS

COFACTOR: |(F(X)-L)/(X-A)| OR |X*3 + 2X*2 + 2X + 8|

ENTER 0  OR THE LINE NUMBER YOU WISH CLARIFIED
⎕:
      2
IN STEP 2  WE ARE DETERMINING

THE DOMAIN OF X  ;  THE CONSTANT IN F(X)-L=2-18=¯16

ENTER 0  OR THE LINE NUMBER YOU WISH CLARIFIED
⎕:
      0
      SUPERGRAPH
INSTRUCTIONS? YES

YOU MAY ENTER ANY EXPRESSION USING + - × ÷ * |  SIN  COS  TAN
AND I WILL PRODUCE A GRAPH OF THE EXPRESSION.  STANDARD RULES
OF ALGEBRA ARE USED TO DETERMINE THE ORDER OF EVALUATION.

F(X) = .5×|2X+|2X+4||
FOR WHAT VALUES OF X ARE WE TO EVALUATE THE FUNCTION
⎕:
      STEP ¯4 4 .5
```



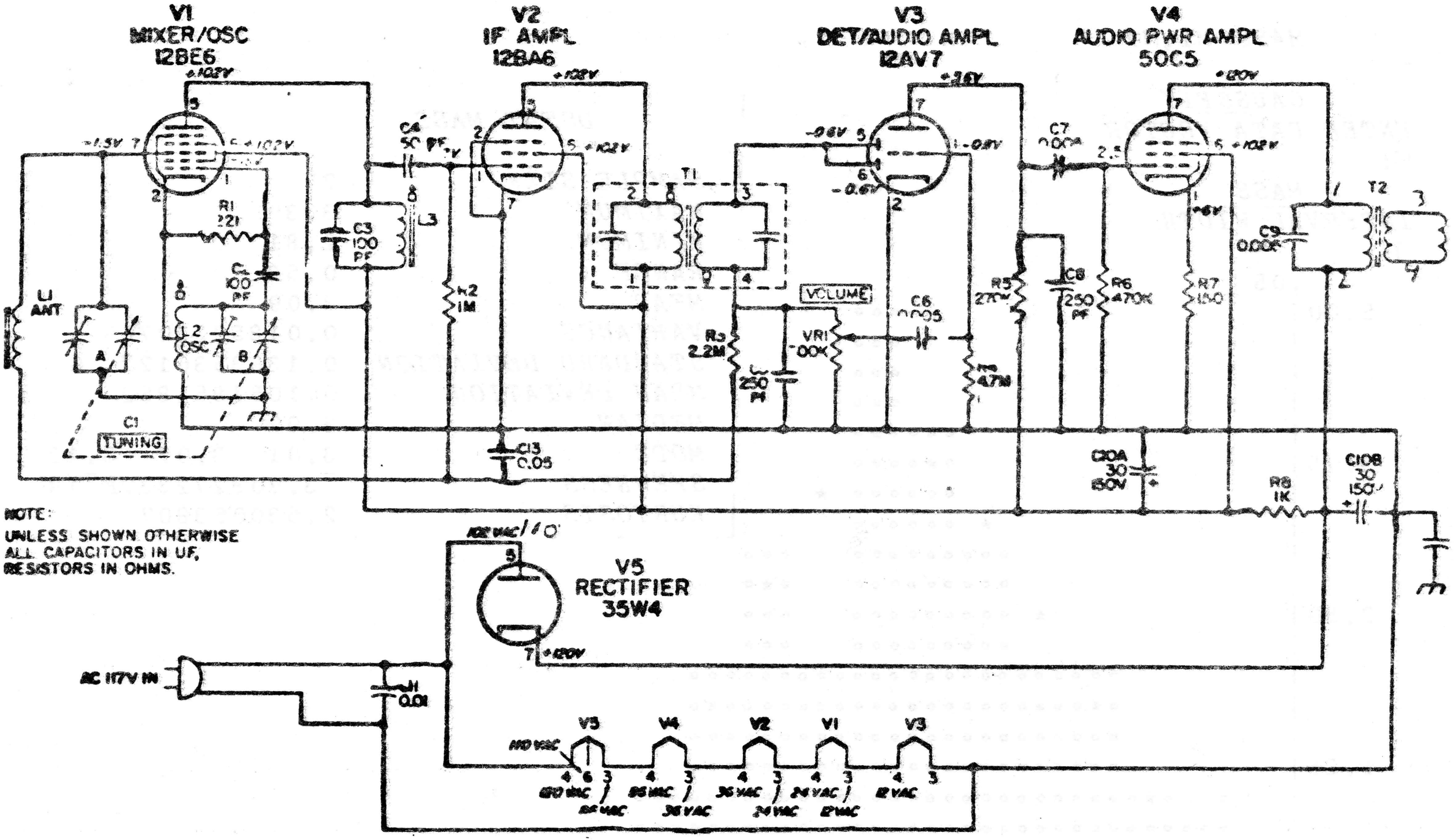
```
DO YOU WANT A TABLE OF THE X AND F(X) VALUES? YES
  ¯4                     2
  ¯3.5                   2
  ¯3                     2
  ¯2.5                   2
  ¯2                     2
  ¯1.5                   1
  ¯1                     0
  ¯0.5                   1
  0                      2
  0.5                    3
  1                      4
  1.5                    5
  2                      6
  2.5                    7
  3                      8
  3.5                    9
  4                     10
```

```
      )LOAD 1234 ZPLOT
SAVED  20.01.58 11/14/72

      ZPLOT

F(X) = (EXP X - X SIN X)(X*2 -8X + 16)
WHAT IS THE DOMAIN OF THE REAL VALUES?
⎕:
      STEP ¯5 6 .2
WHAT IS THE DOMAIN OF THE IMAGINARY VALUES?
⎕:
      STEP ¯2 5 .25

    /------\\\\\\||||||//////-----\\\\\||||||/////----\\\\\| ←5ι
    /------\\\\\\\|||||//////-----\\\\\||||||/////----\\\\||
    /-------\\\\\\||||||/////-----\\\\\\|||||/////---\\\\|||
    //------\\\\\\||||||/////-----\\\\\\|||||/////---\\\||||
    //------\\\\\\||||||/////------\\\\\|||||/////---\\||||| ←4ι
    //-------\\\\\\|||||//////-----\\\\\|||||//////--\||||||
    ///------\\\\\\||||||/////-----\\\\\|||||//////////|////
    ///------\\\\\\||||||/////-----\\\\\|||||///////////////
    ///-------\\\\\\|||||/////-----\\\\\|||||/////////////// ←3ι
    ////------\\\\\\||||||/////----\\\\\|||||//////////////-
    ////-------\\\\\\|||||/////----\\\\||||||/////////------
    /////------\\\\\\||||||////----\\\\|||||////////--------
    /////-------\\\\\\|||||/////---\\\\|||||///////--------- ←2ι
    //////------\\\\\\||||||////---\\\||||||//////----------
    //////-------\\\\\\|||||/////--\\\|||||//////-------\\\\
    ///////------\\\\\\||||||//////\|||||||/////------\\\\\\
    ////////-----\\\\\\\|||||//////||||||||/////----\\\\\\\\ ←1ι
    ////////------\\\\\\\|||||/////|||||||//////---\\\\\\\||
    /////////-----\\\\\\\||||||///||||||||//////--\\\\||||||
    //////////----\\\\\\\\|||||///||||||||///////-\|||||||||
    ∨∨∨∨∨∨∨∨∨∨∧∧∧∧∧∧∧∧∧∧∧∧∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨∨○∨∨∨∨∨∨∨∨∨∨ ←0ι
    ||||||||||\\\\--------/////|||////////|||||||\-/////////
    |||||||||\\\\\-------//////|||////////||||||\\----//////
    ||||||||\\\\\\-------/////|||||///////||||||\\\-------//
    ||||||||\\\\\-------/////||||||////////|||||\\\\-------- ←¯1ι
    |||||||\\\\\\------//////||||||-///////|||||\\\\\\------
    ||||||\\\\\\\------/////|||||\\---/////||||||\\\\\\\----
    ||||||\\\\\\------//////||||\\\---//////||||||\\\\\\\\\\
    |||||\\\\\\\------/////|||||\\\----/////|||||||\\\\\\\\\ ←¯2ι
    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑
   -5.  -4.  -3.  -2.  -1.   0.   1.   2.   3.   4.   5.   6.
```

$$f(x) = (e^{x} - x \sin x)(x^2 - 8x + 16)$$

```
      )LOAD 1234 LEASTFIT
SAVED  11.55.38 12/12/72

      M1←2.81 2.86 3.31 3.26 3.22 2.91 2.98 3.19 3.17 3.16 3.00 3.01 3.24 3.0
      M2←2.93 3.11 3.12 2.96 3.39 3.20 3.10 3.06 3.09 3.07 3.08 3.07 3.12

      MASS←M1,M2

      GAUSSFIT
ENTER DATA VECTOR
⎕:
      MASS
INTERVAL WIDTH
⎕:
      .05
5.00|                     ooo
    |                     ooo
    |                     ooo
    |                     ooo
    |                  oooo*o
3.75|                  o*oooo
    |                  oooooo *
    |                * oooooo
    |               ooooooooo   ooo
    |               ooooooooo   o*o
2.50|             * ooooooooo   ooo
    |               ooooooooo   ooo
    |         oooooooooooooooooooooooo
    |         o*oooooooooooooooooooo*o
    |         oooooooooooooooooooooooo
1.25|         oooooooooooooooooooooooo
    |   oooo*oooooooooooooooooooooooooo*o
    |   ooooooooooooooooooooooooooooooooo
    |   o*ooooooooooooooooooooooooooooooo *  *
    | * ooooooooooooooooooooooooooooooooo
        ↑     ↑     ↑     ↑     ↑     ↑
       2.810 2.910 3.010 3.110 3.210 3.310   ×E0

DO YOU WANT A TABLE OF VALUES?  YES
  INTERVAL CENTER    FREQUENCY          EXPECTED FREQUENCY
       .81               1               0.4831319234
      2.86               1               0.9660180035
      2.91               2               1.667516917
      2.96               2               2.500424743
      3.01               3               3.307342057
      3.06               4               3.844173135
      3.11               5               3.885170268
      3.16               2               3.447080891
      3.21               3               2.669859935
      3.26               2               1.827862884
      3.31               1               1.09026796
      3.36               0               0.5613267593
```

```
      DSTAT MASS

SAMPLE SIZE            27
MAXIMUM                3.39
MINIMUM                2.81
RANGE                  0.58
MEAN                   3.09
VARIANCE               0.01859230769
STANDARD DEVIATION     0.1363536127
MEAN DEVIATION         0.1051851852
MEDIAN                 3.09
MODE                   3.01  3.07  3.12
SKEWNESS               ¯3.908272332E¯14
KURTOSIS               2.530063807
```


V1
MIXER/OSC
12BE6
V2
IF AMPL
12BA6
V3
DET/AUDIO AMPL
12AV7
V4
AUDIO PWR AMPL
50C5
V5
RECTIFIER
35W4
L1 ANT
L2 OSC
L3
C1 TUNING
C3 100 PF
C4 50 PF
C6 .0005
C7 .005
C8 250 PF
C9 0.005
C10A 30 150V
C10B 30 150V
C11 0.01
C13 0.05
R1 22K
R2 1M
R3 2.2M
R4 4.7M
R5 270K
R6 470K
R7 150
R8 1K
VR1 500K
VOLUME
T1
T2
AC 117V IN
NOTE:
UNLESS SHOWN OTHERWISE
ALL CAPACITORS IN UF,
RESISTORS IN OHMS.

```
              )LOAD 1234 ELECTRONICS
SAVED   12.41.00 01/04/73

              TROUBLE
DO YOU WANT INSTRUCTIONS?  NO

NO AUDIO ON LOW BAND
NOISY AUDIO NEAR TOP OF HIGH BAND
SQUELLING ON HIGH BAND
⎕:
              SIGNAL TUBE 3 PIN 5
  NO    SIGNAL
⎕:
              VOLTAGE TUBE 3 PIN 5
 -4.0   VOLTS
⎕:
              REPLACE TUBE 3
 ***TROUBLE PERSISTS***
⎕:
              RESISTANCE RES 4
4.7M    OHMS
⎕:
              REPLACE CAP 13
 ●●●TROUBLE GONE●●●
WANT ANOTHER PROBLEM?  NO
```

TOTALS

624 INSTRUCTIONAL SEGMENTS

IN

37 SUBJECT AREAS

308 HOURS OF INSTRUCTION

DEVELOPED BY

85 DIFFERENT FACULTY MEMBERS

FALL 1972

2000 STUDENTS

ACCUMULATED

20000 HOURS

DURING

21000 TERMINAL SESSIONS

SUBJECT AREA	SEGMENTS	TIME
ACCOUNTING	8	3:35
ALGEBRA	37	20:10
ANATOMY	20	10:00
ARITHMETIC	44	19:30
ART	1	:05
BIOLOGY	16	6:10
BUSINESS LAW	14	9:45
BUSINESS	1	1:00
CAI	3	:40
CALCULUS	42	15:05
CHEMISTRY	31	11:25
COMPUTERS	13	7:45
ECONOMICS	7	8:05
ELECTRONICS	5	2:15
ENGINEERING	1	:50
ENGLISH	38	20:30
FINITE MATH	9	3:20
GEOGRAPHY	2	1:00
GEOLOGY	1	:30
GEOMETRY	7	2:00
GERMAN	3	1:30
GRAPHIC ARTS	3	1:30
LIBRARY	1	:30
LOGIC	5	12:20
MATH	1	:20
MUSIC	1	:40
PHOTOGRAPHY	1	:30
PHYSICS	28	11:25
PLOTTING	6	2:40
POLICE SCI.	31	9:20
POLITIC SCI.	17	9:00
PSYCHOLOGY	3	2:00
SEC. SCI.	8	3:50
SLIDE RULE	14	6:30
SPANISH	193	99:30
STATISTICS	8	2:15
TRIGONOMETRY	1	:20

WHY APL FOR CAI?

I. APL IS A NOTATION NOT A COMPUTER LANGUAGE.

A. IT WAS DESIGNED TO CONVEY IDEAS IN THE CLASSROOM RATHER THAN DRIVE A COMPUTER THROUGH CALCULATIONS.

B. THE COMPUTER IMPLEMENTATION CLOSELY FOLLOWS THE NOTATION AND THEREFORE IS NOT LIMITED BY THE HARDWARE CONCEPTS OF FIRST AND SECOND GENERATION VON NEUMANN MACHINES.

II. APL IS THE ONLY AVAILABLE TELEPROCESSING SYSTEM THAT WILL SUPPORT THE CAI NEEDS OF THE COMMUNITY COLLEGE.

A. DRILL & PRACTICE SEGMENTS.

B. PROBLEM SOLVING COMPUTATIONAL MODE.

C. TUTORIAL WITH MULTIPLE BRANCHING.

D. SIMULATION

III. APL IS THE RICHEST LANGUAGE AVAILABLE.

A. THERE ARE OVER 70 PRIMITIVE FUNCTIONS AND 3 OPERATORS AS OPPOSED TO LESS THAN 1/2 DOZEN PRIMITIVES FOR BASIC, FORTRAN ETC.

B. EXTENSION OF DATA TYPES TO TENSORS (SCALARS, VECTORS, MATRICES, ARRAYS ETC.) ELIMINATES THE NEED FOR LOOPING IN MANY CASES AND REDUCES THE PROGRAMING TO SIMPLE DESCRIPTIONS OF THE PROBLEM.

IV. APL SUPPORTS THE CREATION OF SPECIALIZED CAI LANGUAGES.

A. FUNCTION CALL STRUCTURE ALLOWS FOR EASY COMMUNICATION AND TRANSFER OF DATA BETWEEN FUNCTIONS.

B. FUNCTIONS WRITTEN FOR ONE C A I SEGMENT MAY BE EASILY COPIED AND USED FOR ANOTHER SEGMENT.

C. UNDER PROGRAM CONTROL, IT IS EASY TO TURN THE SYSTEM OVER TO THE CALCULATOR MODE IF REQUESTED BY THE STUDENT.

V. APL ALSO SUPPORTS AND MEETS STUDENTS NEEDS FOR A PROGRAMING LANGUAGE.

A. THE LANGUAGE IS EASY TO LEARN.

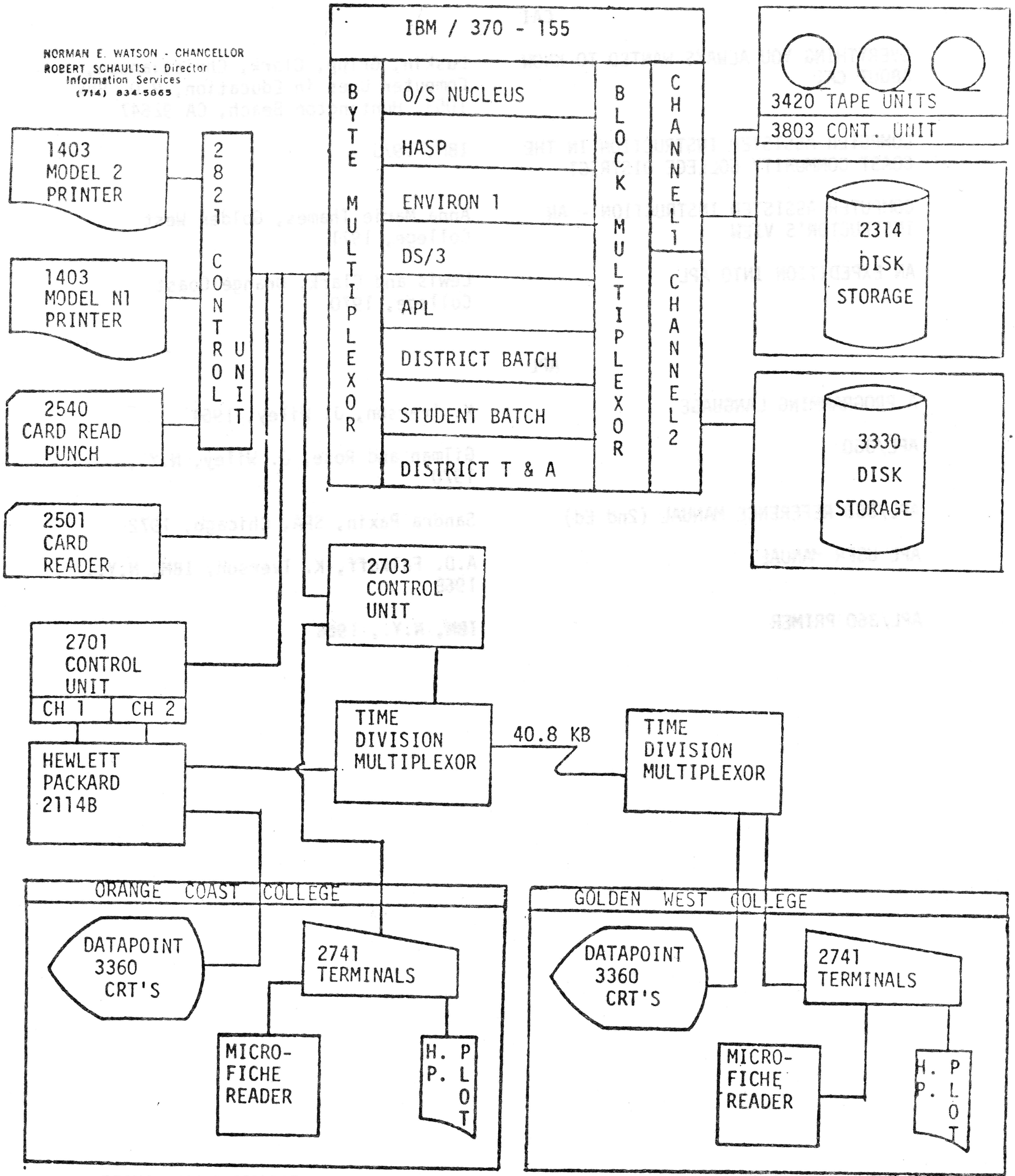
B. STUDENT MAY PRETEST EACH STEP OF PROGRAM BEFORE ENTERING IT.

C. STUDENT IS NOT BOGGED DOWN WITH FLOWCHARTING, DATA STRUCTURES, LOOPING TECHNIQUES, AND OTHER TRIVIA AND IS ALLOWED TO SOLVE PROBLEMS IN A NATURAL WAY.

D. INPUT/OUTPUT IS EASY TO HANDLE AND REQUIRES ONLY PRIMITIVE FUNCTIONS.

Coast Community College district
Information Services
2701 FAIRVIEW ROAD · COSTA MESA · CALIFORNIA 92626

NORMAN E. WATSON - CHANCELLOR
ROBERT SCHAULIS - Director
Information Services
(714) 834-5865


IBM / 370 - 155
BYTE MULTIPLEXOR
O/S NUCLEUS
HASP
ENVIRON 1
DS/3
APL
DISTRICT BATCH
STUDENT BATCH
DISTRICT T & A
BLOCK MULTIPLEXOR
CHANNEL 1
CHANNEL 2
3420 TAPE UNITS
3803 CONT. UNIT
2314 DISK STORAGE
3330 DISK STORAGE
1403 MODEL 2 PRINTER
1403 MODEL N1 PRINTER
2540 CARD READ PUNCH
2501 CARD READER
2821 CONTROL UNIT
2703 CONTROL UNIT
2701 CONTROL UNIT
CH 1
CH 2
HEWLETT PACKARD 2114B
TIME DIVISION MULTIPLEXOR
40.8 KB
TIME DIVISION MULTIPLEXOR
ORANGE COAST COLLEGE
DATAPOINT 3360 CRT'S
2741 TERMINALS
MICRO-FICHE READER
H. P. PLOT
GOLDEN WEST COLLEGE
DATAPOINT 3360 CRT'S
2741 TERMINALS
MICRO-FICHE READER
H. P. PLOT

ORANGE COAST COLLEGE
COSTA MESA

1972 - 1973

GOLDEN WEST COLLEGE
HUNTINGTON BEACH

REFERENCES

CAI

EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT CAI	Luskin, Gripp, Clark, Christianson, Computer Uses in Education, P.O. Box 1053, Huntington Beach, CA 92647
COMPUTER ASSISTED INSTRUCTION IN THE COAST COMMUNITY COLLEGE DISTRICT	IBM, 1973
COMPUTER ASSISTED INSTRUCTION - AN INSTRUCTOR'S VIEW	Anna Marie Thames, Golden West College, 1971
AN EXPEDITION INTO APL	Lewis and Clark, Orange Coast College, 1970

APL

A PROGRAMMING LANGUAGE	K. Iverson, J. Wiley, 1964
APL/360	Gilman and Rose, J. Wiley, N.Y., 1970
APL/360 REFERENCE MANUAL (2nd Ed)	Sandra Paxin, SRA, Chicago, 1972
APL USER MANUAL	A.D. Falkoff, K. Iverson, IBM, N.Y., 1968
APL/360 PRIMER	IBM, N.Y., 1968

A Table-Drive Program to Play a Game of Logical Deduction

S. Charmonman[a] and D. K. Reed[b]

Abstract. A game of logical deduction called Pico-Fumi is described formally and simulated in APL. Observations are made of the results of the simulations to set up a computer strategy. This strategy is in the form of a table of keys of previous results and actions for the next moves. Simplification is made of the table and an APL function written to use it. If the computer's opponent supplies inconsistent information, the computer is also programmed to find out where the inconsistency occurred. The program has been successfully used as a first step to get some high school students to become interested in computer sciences and computer applications.

Introduction. The senior author has found that game-playing programs attract and hold more attention from high school students than most any other programs such as the conventional mathematical ones. Various groups of high school students have been and will be invited by the University of Missouri to visit the campus. Therefore it is highly desirable to develop a package of programs to play games of interest to high school students. Once they get a hold of using the computer then they can be led into other areas of computer sciences and computer applications.

As an example of games of interest to school children, the junior author suggested the game of Pico-Fumi or Pico-Foumi[5]. It was stated in the May 1972 issue of the Arithmetic Teacher[1] that a school principal had played this game in the Philadelphia area. A generalization of the game has also been proposed[2].

---

[a] Associate Professor and Director of Graduate Studies, Department of Computer Science, University of Missouri-Columbia.

[b] Teaching Assistant, Department of Computer Science, University of Missouri-Columbia.

Pico-Fumi is a two-player game of logical deduction. The objective is for each player to use logically deduced guesses to pinpoint a two-digit number picked by his opponent. Play begins with each player selecting a number with two distinct digits. Zero may be used in either the first or the second position, e.g., 05 is a valid number. The play continues with players taking turns guessing his opponent's number. To each guess, the opponent must respond by giving

a) the number of Pico, one Pico for each digit in the guess which is correct but not in the correct position, and

b) the number of Fumi, one Fumi for each digit in the guess which is correct and in the correct position.

For example, suppose Player A's chosen number is 37. If Player B's guess is 59 he gets zero Pico and zero Fumi. If Player B's guess is 72 he gets one Pico for 7 and no Fumi. If Player B's guess is 67 he gets no Pico and one Fumi for 7. If Player B's guess is 73 he gets two Pico and no Fumi. If Player B's guess is 37 he gets no Pico and two Fumi and wins the game.

In general, the player who guesses his opponent's number right first, wins the game.

APL Description of the Game. The game of Pico-Fumi may be described

---

The numerical superscripts refer to the items in the list of references.

formally and simulated in APL function named PICOSIMULATION as shown in Figure 1. No strategy is used in the simulation. As a matter of fact, the simulation will be used to set up the computer's strategy.

Computer Strategy. In a game of perfect information[6], such as Tic Tac Toe[3] and Nim[4], it is possible to program the computer to be the best player who never loses the game. However, Pico-Fumi is neither a game of perfect information nor a game of 100% chance. So, the best the computer can do is to be a good player.

A strategy for the computer to be a good player may be drawn up as follows:

First, a logical deduction and/or an observation of the results of several runs of PICOSIMULATION may be used to state that

a) the sum of Pico and Fumi must be less than or equal to two because there are only two digits, and

b) the combination one-one does not occur because if one digit of the guess gives one Fumi (i.e., it is the correct digit and at the correct position), the other digit must be either incorrect (giving zero-one) or correct and at the correct position (giving zero-two). Thus, there are only five valid combinations of Pico-Fumi namely zero-zero, zero-one, one-zero, zero-two and two-zero.

Let D be the list of all possible digits for the computer to use in its guesses. At the start of the game, with)*ORIGIN* 0,

```
[1]             D ← ⍳10
```

```
    ∇ PICOSIMULATION
[1]    D←'0123456789'
[2]   PICKNO:'THE NUMBER CHOSEN BY PLAYER 0 IS ';N0←D[2?10];', AND'
[3]    'THE NUMBER CHOSEN BY PLAYER 1 IS ';N1←D[2?10];'.'
[4]    N←N0,[¯0.5] N1
[5]    P←1
[6]   TAKETURN:P←~P
[7]   GUESS:'THE GUESS MADE BY PLAYER ';P;' IS ';G←D[2?10]
[8]   RESPONSE:'THE RESPONSE FROM PLAYER ';~P;' IS ';+/G=⌽N[~P;];' PICO
       AND ';C←+/G=N[~P;];' FUMI.'
[9]    →(C≠2)/TAKETURN
[10] OVER:'PLAYER ';P;' HAS WON THIS GAME.'
    ∇


      PICOSIMULATION

THE NUMBER CHOSEN BY PLAYER 0 IS 21, AND
THE NUMBER CHOSEN BY PLAYER 1 IS 49.
THE GUESS MADE BY PLAYER 0 IS 82
THE RESPONSE FROM PLAYER 1 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 1 IS 49
THE RESPONSE FROM PLAYER 0 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 0 IS 46
THE RESPONSE FROM PLAYER 1 IS 0 PICO AND 1 FUMI.
THE GUESS MADE BY PLAYER 1 IS 27
THE RESPONSE FROM PLAYER 0 IS 0 PICO AND 1 FUMI.
THE GUESS MADE BY PLAYER 0 IS 84
THE RESPONSE FROM PLAYER 1 IS 1 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 1 IS 19
THE RESPONSE FROM PLAYER 0 IS 1 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 0 IS 42
THE RESPONSE FROM PLAYER 1 IS 0 PICO AND 1 FUMI.
THE GUESS MADE BY PLAYER 1 IS 67
THE RESPONSE FROM PLAYER 0 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 0 IS 46
THE RESPONSE FROM PLAYER 1 IS 0 PICO AND 1 FUMI.
THE GUESS MADE BY PLAYER 1 IS 49
THE RESPONSE FROM PLAYER 0 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 0 IS 56
THE RESPONSE FROM PLAYER 1 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 1 IS 65
THE RESPONSE FROM PLAYER 0 IS 0 PICO AND 0 FUMI.
THE GUESS MADE BY PLAYER 0 IS 54
THE RESPONSE FROM PLAYER 1 IS 1 PICO AND 0 FUMI.

PICOSIMULATION[8]
```

Figure 1: Simulation of Pico-Fumi

Let N be the computer's guess. The first guess may be chosen at random from D,

```
[2]                    N ← D[2?ρD]
```

The combination zero-zero indicates that both of the digits in the guess do not occur in the opponent's number. So they can be dropped from the list of possible digits,

```
[3]                    D ← (~D∊N)/D
```

The combination zero-two means that the computer has won the game.

The combination two-zero is obtained when both digits in the guess are correct, but not at the correct position. The computer will win in its next guess by reversing the two digits.

```
[4]                    N ← ⌽N
```

The combinations zero-one and one-zero indicate that one digit of the guess is correct and at the correct position in the case of zero-one, but not at the correct position in the case of one-zero. The computer's action will depend on what previous Pico-Fumi combinations it has obtained so far. If it has already pin-pointed the other digit, it can win in the next guess. Otherwise it has to make more guesses to find the other digit.

Basic Strategy. The computer uses two previous sets of Pico-Fumi values to determine its action. Thus, the action at the (I+1)th stage depends on the best guess prior to the Ith guess and the Ith guess itself. The strategy is summarized in Figure 2. As stated

Row No.	Pico-Fumi Value				Action No.	Sub-Actions	Next Guess Leads to
	P_B	F_B	P_I	F_I			
1	0	0	0	0	0	[3][2]	rows 2 to 5
2	0	0	0	1	1	[3][5][6]	rows 6 to 9
3	0	0	0	2	2	computer wins	
4	0	0	1	0	3	[4][3][5][6]	rows 6 to 9
5	0	0	2	0	4	[4]	computer wins
6	0	1	0	0	5	[3][7]	rows 7 or 8
7	0	1	0	1	6	[3][8]	rows 7 or 8
8	0	1	0	2	2	computer wins	
9	0	1	1	0	7	[9]	computer wins
10	0	1	2	0	8	error	
11	1	0	0	0	8	error	
12	1	0	0	1	8	error	
13	1	0	0	2	8	error	
14	1	0	1	0	8	error	
15	1	0	2	0	8	error	

Figure 2: A Table To Drive Pico-Fumi Playing Program

before, only five combinations of Pico-Fumi values are possible. The combination zero-two terminates the game right away. The combination two-zero will allow the computer to win the very next move. Therefore only three combinations of P_B and F_B need be considered as the Pico-Fumi values for the best previous guess. The size of the table is thus 15-by-5 as shown in Figure 2.

On the first row of the table, both sets of Pico-Fumi values are zero and the action is to update the list of digits as in the APL assignment statement [3] and pick two remaining digits as in [2].

On the second row of the table when a zero-zero-zero-one occurs, one digit of the guess is correct and at the correct position. The action should be to update the list as in [3]. Keep the currect guess somewhere, say in M,

[5] $M \leftarrow N$

Then try to find which of the two digits in N is the good one by using one of them at a time in the subsequent guesses.

[6] $N[T \leftarrow ?2] \leftarrow D[?\rho D]$

With zero-zero-one-zero key on the fourth row of the table, the computer is told that one of the digits is correct but not at the correct position. It should find out which digit is the good one. One way is to reverse the present guess as in [4] to get into the second row at no extra move.

On the sixth row, both the digits chosen from the previous guess and the new digit from D do not occur in the opponent number.

The computer should update the list and go back to the other digit in M.

```
                         N ← M
                    N[~T] ← D[?ρD]
```

or, in one line,

```
[7]         N ← (M×~T1)+ D[?ρD]× T1 ← 0 1 = T←~T
```

The succeeding guesses will give zero-one until the computer wins.

On the seventh row, the digit chosen from the previous guess is correct and at the correct position, but the new digit from D is not. So the computer updates D as in [3] and chooses a new digit,

```
[8]                     N[T] ← D[?ρD]
```

The subsequent guesses will loop over the seventh row until the computer wins.

On the ninth row, the wrong digit was chosen from M in the second or the fourth row. The new digit picked at random earns a Pico and is therefore correct but not in the correct position. The computer can win in its next move by taking the other digit from M and the new digit from N,

```
[9]           N ← (M×T1) + (⌽N)× ~T1 ← T = 0 1
```

The remaining Pico-Fumi combinations have been voided by the previous actions. In particular, the action in the second or fourth row voids the combinations in the tenth to fifteenth rows and the action in the third, fifth, eighth or ninth row terminates the game with the computer winning.

The Table in Figure 2 may be used to drive a program to play Pico-Fumi but a more efficient table is possible as shown in the next section.

Improved Strategy. An improvement to the table in Figure 2 may be made to reduce the size of the table and thus the computation time. The four-digit key, P_B, F_B, P_I and F_I may be reduced to a single base-3 number,

$$KEY \leftarrow 3 \perp P_B, F_B, P_I, F_I$$

The third and eighth rows may be eliminated from the table by checking outside the table whether or not the computer has won the game. The action number may be represented by the position of the key.

Thus, the table may be reduced to a vector of size seven as shown in the column KEY in Figure 3.

Element No.	Key	Action	Next guess will lead to
0	0	[3][2]	Key 1 or 3 or wins
1	1	[3][5][6]	Key 9 or 10 or 12 or wins
2	3	[3][4][6]	Key 9 or 10 or 12 or wins
3	6	[4]	Computer wins
4	9	[3][7]	Key 10 or wins
5	10	[3][8]	Key 10 or wins
6	12	[9]	Computer wins

Figure 3: Improved Table of Computer Strategy

Computer Implementation. The APL function at the heart of the Pico-Fumi playing program is based on the table in Figure 3 and named CALLTAB as shown in Figure 4. It gives the computer's next educated guess. If CALLTAB detects that the computer's opponent has been inconsistent in supplying the Pico-Fumi information it will set N to zero. The main program named PICO will then print the message and produce the correct information. If the opponent tells the computer his number at this point of the game, the computer will also print the place or places where the opponent was inconsistent. The checking is done in the APL function PICOCHECK.

The computer's opponent has to make logical deductions to be a good player. He may set up the table of his guesses and Pico-Fumi information by himself or ask the machine to print it for him. The function to do the printing is called PICOPRINT and is shown in Figure 4.

A few sample games are given in Figure 5.

The Pico-Fumi playing program has been successfully used as a first step to get some high school students to become interested in computer sciences and computer applications.

Acknowledgments. This work was started as a part of a larger project on Efficient Algorithms and APL supported by the National Foundation under grant No. NSF GJ426.

```
     ∇ CALLTAB KEY;S;ACT;T1
[1]     →(3≠ACT← 0 1 3 6 9 10 12 ⍳KEY)/CONT
[2]    L3:→0,(W←6),N←⌽N
[3]    CONT:→((0=S←⍴D←(~D∊N)/D)∨((0≠FREE)∧KEY≠10))/L6
[4]     →(L0,L1,L2,L3,L4,L5,L6,L7)[ACT]
[5]    L0:→0,N←D[2?S]
[6]    L2:FREE←¯1+1↑⍴CG←CG,[0] N←⌽N
[7]     CPF←CPF,[0] 0 1
[8]    L1:M←N
[9]     →0,(W←1),N[T←?2]←D[?S]
[10]   L4:→0,(W←2),N←(M×~T1)+D[?S]×T1←(KP← 0 1)=T←~T
[11]   L5:→0,(W←2),N[T]←D[?S]
[12]   L6:→0,(W←6),N←(M×T1)+(⌽N)×~T1←T= 0 1
[13]   L7:N←,0
     ∇


     ∇ G PICOPRINT PF;T
[1]     ''
[2]     '          GUESS        FIRST        SECOND        PICO          FUMI'
[3]     '           NO.         DIGIT        DIGIT'
[4]     ''
[5]     (60⍴(11⍴0),1)\ 1 0 ↓DG[(⍳1↑⍴G),G,PF]
     ∇


     ∇ PICOCHECK;R;T
[1]     ''
[2]     NAME̲;', PLEASE TYPE YOUR NUMBER.'
[3]     R←DG⍳(R≠' ')/R←⍞
[4]     →(≠/R)/OK
[5]     →0,0⍴⎕←'THE TWO DIGITS MUST NOT BE THE SAME.'
[6]    OK:CPF2←⍉(+/[0] 1 0 0 ⍉⌽T),[¯0.5]+/[0] 1 0 0 ⍉T←CG∘.=R
[7]     '
        YOUR RESPONSES TO MY GUESSES SHOULD HAVE BEEN
        '
[8]     CG PICOPRINT CPF2
[9]     NAME̲;', YOU MADE INCORRECT RESPONSES AT LINE NOS. ';(∨/CPF≠CPF2)/⍳1
        ↑⍴CPF
     ∇
```

Figure 4: APL Functions to Play Pico Fumi

```
     ∇ PICO
[1]    'PLEASE ENTER YOUR NAME.'
[2]    NAME̲←⍞
[3]    CWON←YWON←0
[4]    DG←'0123456789'
[5]    ''
[6]    'ARE YOU FAMILIAR WITH THE RULES OF THE GAME, ';NAME̲;'?
       '
[7]    →('Y'=1↑⍞)/PLAY
[8]    '[1] EACH PLAYER CHOOSES A NUMBER WITH TWO DISTINCT DIGITS.'
[9]    '[2] THEN EACH PLAYER TAKES TURNS IN GUESSING THE OPPONENT''S NUMBE
       R.'
[10]   '[3] THE PLAYER WHO GUESSES CORRECTLY FIRST WINS THE GAME.'
[11]   '[4] A PLAYER MUST RESPOND TO THE OPPONENT''S GUESS BY GIVING:'
[12]   '          A. THE NUMBER OF PICO (ONE PICO FOR EACH DIGIT IN THE'
[13]   '     GUESS WHICH IS CORRECT BUT NOT IN THE CORRECT POSITION), AND'
[14]   '          B. THE NUMBER OF FUMI (ONE FUMI FOR EACH DIGIT IN THE'
[15]   '     GUESS WHICH IS CORRECT AND IN THE CORRECT POSITION).
       '
[16] PLAY:'O.K. PLEASE SET UP YOUR NUMBER WITH TWO DISTINCT DIGITS'
[17]   '      --BUT  DON''T LET ME KNOW IT.
       '
[18]   OPG←CG←OPF←CPF← 1 2 ⍴KEY←FREE←W←0
[19]   N←D[2?⍴D←⍳10]
[20]   NUM←2?10
[21]   'WOULD YOU LIKE TO HAVE THE FIRST MOVE? YES OR NO?'
[22]   →('N'=1↑⍞)↓NEXT,COMTURN
[23] OPP:' WOULD YOU LIKE TO SEE A TABLE OF YOUR GUESSES, PICO AND FUMI?
       '
[24]   →('N'=1↑⍞)/NEXT
[25]   OPG PICOPRINT OPF
[26] NEXT:NAME̲;', IT''S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.'
[27] TRY:→((10∊R)∨=/R←'0123456789'⍳R←(R≠' ')/R←⍞)/ERR
[28]   OPG←OPG,[0] R
[29]   →(∧/NUM=R)/END
[30]   ''
[31]   ' YOU GOT ';P←+/NUM=⌽R;' PICO AND ';F←+/NUM=R;' FUMI.
       '
[32]   OPF←OPF,[0] P,F
[33]   →(1=1↑⍴CG)/COMTURN
[34]   CALLTAB KEY
[35]   →(1=⍴N)/OPPERR
[36] COMTURN:CG←CG,[0] N
[37] CGUESS:NAME̲;', IS YOUR NUMBER ';DG[N];'? YES OR NO?'
[38]   →(ANS←'N'=1↑⍞)/N1
[39]   '
       I WON THIS GAME!
       '
[40]   CWON←CWON+1
[41]   'WOULD YOU LIKE TO SEE MY NUMBER?'
```

Figure 4: APL Functions to Play Pico Fumi (Continued)

```
[42]   →('N'=1↑⍞)/OUT
[43]   'O.K. IT IS ';NUM
[44]   →OUT
[45]  N1:'HOW MANY PICO DID I GET?'
[46]   CP←⎕
[47]   ''
[48]   'HOW MANY FUMI DID I GET?'
[49]   TP1←10⊥TPF←CP,CF←⎕
[50]   →((2≥+/TPF)∧(1≠×/TPF)∧((W≠1)∨(W=1)∧TP1∊ 0 1 10)∧((W≠2)∨(W=
       2)∧TP1=1)∧(W≠6)∨(W=6)∧ANS≠1)/OKPF
[51]   CPF←CPF,[0] TPF
[52]   →OPPERR
[53]  OKPF:CPF←CPF,[0] TPF
[54]   →(0=FREE)/OKKEY
[55]   KEY←3⊥ 0 1 ,, ¯1 2 ↑CPF
[56]   →GO
[57]  OKKEY:KEY←3⊥, ¯2 2 ↑CPF
[58]  GO:→(1≠1↑⍴OPG)↓NEXT,OPP
[59]  ERR:''
[60]   NAME̲;' YOUR GUESS IS NOT A NUMBER WITH TWO DISTINCT DIGITS.'
[61]   'PLEASE TRY AGAIN'
[62]   →TRY
[63]  OPPERR:''
[64]   NAME̲;', YOU HAVE GIVEN ME INCONSISTENT PICO-FUMI VALUES'
[65]   'AS SHOWN IN THE TABLE WITH MY GUESSES BELOW:'
[66]   CG PICOPRINT CPF
[67]   →(0=FREE)/NOTFREE
[68]   '
       N̲O̲T̲E̲: MY GUESS NO. ';FREE;' WAS NOT ACTUALLY MADE BUT LOGICALLY
       DEDUCED FROM THE IMMEDIATELY PRECEEDING GUESS.'
[69]  NOTFREE:'
       WOULD YOU LIKE TO SEE WHERE THE ERRORS OCCURED?'
[70]   →('N'=1↑⍞)/NOSIR
[71]   PICOCHECK
[72]  NOSIR:''
[73]   'SHALL WE START THE GAME AGAIN?'
[74]   →('N'=1↑⍞)↓PLAY,OUT1
[75]  END:'CONGRATULATION ';NAME̲;', YOU HAVE WON THIS GAME!'
[76]   YWON←YWON+1
[77]  OUT:''
[78]   'SO FAR, YOU HAVE WON ';YWON;' GAME(S)'
[79]   '       AND I HAVE WON ';CWON;' GAME(S).'
[80]   ''
[81]   '  WOULD YOU LIKE TO PLAY ANOTHER GAME, ';NAME̲;'?'
[82]   →('Y'=1↑⍞)/PLAY
[83]  OUT1:'  HOPE YOU ENJOYED PLAYING PICO-FUMI.  LET''S PLAY AGAIN SOMET
       IME.'
     ∇
```

Figure 4: APL Functions to Play Pico Fumi (Continued)

```
      PICO
PLEASE ENTER YOUR NAME.
SAK

ARE YOU FAMILIAR WITH THE RULES OF THE GAME, SAK?

YES, I THINK SO.
O.K. PLEASE SET UP YOUR NUMBER WITH TWO DISTINCT DIGITS
     --BUT  DON'T LET ME KNOW IT.

WOULD YOU LIKE TO HAVE THE FIRST MOVE? YES OR NO?
NO, WHY DON'T YOU TRY FIRST?
SAK, IS YOUR NUMBER 41? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      0

HOW MANY FUMI DID I GET?
⎕:
      0
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
37

 YOU GOT 0 PICO AND 0 FUMI.

SAK, IS YOUR NUMBER 60? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      2

HOW MANY FUMI DID I GET?
⎕:
      0
 WOULD YOU LIKE TO SEE A TABLE OF YOUR GUESSES, PICO AND FUMI?

NO
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
49

 YOU GOT 2 PICO AND 0 FUMI.

SAK, IS YOUR NUMBER 06? YES OR NO?
YES

I WON THIS GAME!

WOULD YOU LIKE TO SEE MY NUMBER?
NO, DON'T TELL ME YET. I WANT TO PLAY ANOTHER GAME.

SO FAR, YOU HAVE WON 0 GAME(S)
      AND I HAVE WON 1 GAME(S).

  WOULD YOU LIKE TO PLAY ANOTHER GAME, SAK?
YES, I TOLD YOU SO.
```

Figure 5: Sample Game

```
O.K. PLEASE SET UP YOUR NUMBER WITH TWO DISTINCT DIGITS
     --BUT  DON'T LET ME KNOW IT.

WOULD YOU LIKE TO HAVE THE FIRST MOVE? YES OR NO?
NO
SAK, IS YOUR NUMBER 35? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      0

HOW MANY FUMI DID I GET?
⎕:
      0
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
47

 YOU GOT 1 PICO AND 0 FUMI.

SAK, IS YOUR NUMBER 82? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      1

HOW MANY FUMI DID I GET?
⎕:
      1

SAK, YOU HAVE GIVEN ME INCONSISTENT PICO-FUMI VALUES
AS SHOWN IN THE TABLE WITH MY GUESSES BELOW:

         GUESS       FIRST       SECOND       PICO        FUMI
          NO.        DIGIT       DIGIT

          1            3           5           0           0
          2            8           2           1           1

WOULD YOU LIKE TO SEE WHERE THE ERRORS OCCURED?
YES

SAK, PLEASE TYPE YOUR NUMBER.
28

YOUR RESPONSES TO MY GUESSES SHOULD HAVE BEEN


         GUESS       FIRST       SECOND       PICO        FUMI
          NO.        DIGIT       DIGIT

          1            3           5           0           0
          2            8           2           2           0
SAK, YOU MADE INCORRECT RESPONSES AT LINE NOS. 2

SHALL WE START THE GAME AGAIN?
YES
```

Figure 5: Sample Games (Continued)

```
O.K. PLEASE SET UP YOUR NUMBER WITH TWO DISTINCT DIGITS
     --BUT  DON'T LET ME KNOW IT.

WOULD YOU LIKE TO HAVE THE FIRST MOVE? YES OR NO?
YES
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
83

 YOU GOT 0 PICO AND 1 FUMI.

SAK, IS YOUR NUMBER 24? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      0

HOW MANY FUMI DID I GET?
⎕:
      0
 WOULD YOU LIKE TO SEE A TABLE OF YOUR GUESSES, PICO AND FUMI?

NO
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
43

 YOU GOT 0 PICO AND 1 FUMI.

SAK, IS YOUR NUMBER 17? YES OR NO?
NO
HOW MANY PICO DID I GET?
⎕:
      1

HOW MANY FUMI DID I GET?
⎕:
      0
 WOULD YOU LIKE TO SEE A TABLE OF YOUR GUESSES, PICO AND FUMI?

NO
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
13
CONGRATULATION SAK, YOU HAVE WON THIS GAME!

SO FAR, YOU HAVE WON 1 GAME(S)
      AND I HAVE WON 1 GAME(S).

  WOULD YOU LIKE TO PLAY ANOTHER GAME, SAK?
YES
```

Figure 5: Sample Games (Continued)

```
O.K. PLEASE SET UP YOUR NUMBER WITH TWO DISTINCT DIGITS
     --BUT  DON'T LET ME KNOW IT.

WOULD YOU LIKE TO HAVE THE FIRST MOVE? YES OR NO?
N
SAK, IS YOUR NUMBER 59? YES OR NO?
N
HOW MANY PICO DID I GET?
⎕:
      1

HOW MANY FUMI DID I GET?
⎕:
      0
SAK, IT'S YOUR TURN NOW.  PLEASE MAKE YOUR GUESS.
63

 YOU GOT 0 PICO AND 0 FUMI.

SAK, IS YOUR NUMBER 98? YES OR NO?
N
HOW MANY PICO DID I GET?
⎕:
      2

HOW MANY FUMI DID I GET?
⎕:
      0

SAK, YOU HAVE GIVEN ME INCONSISTENT PICO-FUMI VALUES
AS SHOWN IN THE TABLE WITH MY GUESSES BELOW:

         GUESS       FIRST       SECOND        PICO        FUMI
          NO.        DIGIT       DIGIT

          1            5           9            1            0
          2            9           5            0            1
          3            9           8            2            0

NOTE: MY GUESS NO. 2 WAS NOT ACTUALLY MADE BUT LOGICALLY
DEDUCED FROM THE IMMEDIATELY PRECEEDING GUESS.

WOULD YOU LIKE TO SEE WHERE THE ERRORS OCCURED?
Y

SAK, PLEASE TYPE YOUR NUMBER.
86

YOUR RESPONSES TO MY GUESSES SHOULD HAVE BEEN


         GUESS       FIRST       SECOND        PICO        FUMI
          NO.        DIGIT       DIGIT

          1            5           9            0            0
          2            9           5            0            0
          3            9           8            1            0
SAK, YOU MADE INCORRECT RESPONSES AT LINE NOS. 1  2  3
```

Figure 5: Sample Game (Continued)

References

1. Douglas B. Aichele, "Pica-Centro--a game of logic," the Arithmetic Teacher, Vol. 19, No. 5 (May 1972) pp. 359-361.

2. S. Charmonman and D. K. Reed, "Pick-a-Word: a base-26 generalization of Pica-Centro," ~~submitted to~~ accepted by the Arithmetic Teacher.

3. S. Charmonman and J. Sy, "A Non-Losing Tic Tac Toe program," submitted to APL Quote Quad.

4. S. Charmonman and G. Heider, "APL Analysis of a simple binary game," submitted to APL Quote Quad.

5. Robert C. Gessel, Carolyn Johnson, Marty Boren and Charles Smith, "Rainy Day Games," the Arithemtic Teacher, Vol. 19, No. 4 (April 1972), pp. 303-305.

6. E. S. Ventzel, Introduction to the Theory of Games, D. C. Heath and Company, Boston, 1963.

COMMUNICATION, COMPUTERS, AND THE CLASSROOM

by

Leslie N. Davis, Jak Eskinazi and Daniel J. Macero

Department of Chemistry
Syracuse University
Syracuse, New York 13210

INTRODUCTION

Consider a constantly updated cathode ray tube display of Dow Jones averages in an office in Fairbanks, Alaska, or a remote teletype in a laboratory printing pH values as a reaction is simulated in slowed-down time: communication - fast, accurate, and reliable - is what computers are all about. Teachers, businessmen, students, and researchers have a common interest in the vast, and largely untapped, potential of computers for rapid communication of ideas and data.

As teachers and researchers, we have invested much time in developing APL programs optimized for communication. This paper describes some of the ways we have used APL to communicate with our students and with the computer.

WHY APL?

APL is the best available programming language where interactive communication is of high priority.

Knowledge of APL is not necessary for use of an interactive APL program, as we have amply demonstrated (1,2,3). This is important to users who are not concerned at all with programming, i.e. a production manager who wants warehouse stock information, or a French student who wants a vocabulary drill.

The elegance of APL makes it easy to program with. We are all familiar with simple APL statements (i.e. for averaging the components of a vector) which require as much as a dozen or more programming lines in other languages.

Finally, APL incorporates an extensive and specific debugging system, greatly speeding this process as compared to other languages, with a consequent economy of programming time, of particular importance to those of us for whom the programs are not an end in themselves, but a means to an end.

PROGRAMMING WITH CAIR

Successful programs depend not only on the properties of the programming language, but also on the ability of the programmer to develop algorithms maximized for communicability. Such programs will, regardless of the language of conception, adhere closely to the principles we have established for "CAIR in program design" (3), i.e., Consistency, Accessibility, Interactivity, and Real-life conformity, the meanings of which are briefly summarized in the following paragraphs.

The detailed structure of consistent programs furthers design goals systematically, eliminating needless repetition, and imparting a uniqueness to the experience of each program use.

Software and hardware must be readily accessible to users naive in the language mechanistics. Both program design and physical facilities play a part in accessibility.

Truly interactive programs are characterized by flexibility and adaptability in the face of varied and changing user needs, skills, and goals. For example, a student using our program on pH and

Logarithms (2) which begins with an understanding of exponents and moves step by step through complex logarithm calculations to pH and pH-problem solving, could begin at the very beginning if his background in logarithmic manipulations is poor, or could skip the section on logarithms entirely and start right out with the section on pH if his background in logarithms is good. Moreover, at any point in the program it is possible to go back and review a topic as well as skip forward, allowing complete flexibility for the student to determine his own pace or to allow the computer to pace him.

Conformity to real life means program operation is not limited by constraints inherent in the program or computer system. The computer should converse with the user in his native tongue where such interaction is required, or a plot program, for example, should plot to a scale inherent in the data rather than require the data to fit the program, or a data display function should return a display that is logical and consistent with the nature of the material displayed rather than a display limited by the requirements of a specific program language.

The programs described below demonstrate how employment of the CAIR principles results in programs which are effective in efficient communication.

ELEMENTS

The ability of the computer to tirelessly drill students in vocabulary, multiplication, symbols of the chemical elements, or whatever, has recommended it as a technique of choice for mastery of this type of material from the earliest inception of computer assisted instruction (CAI). Use of the APL language and associated hardware for a simple column matching drill, however, can justifiably be criticized as wasteful of computer resources. If the computer cannot exceed in efficiency what is possible with paper, pencil, and a card to cover a column of data with, the computer should not be used for drill purposes.

Consider the set of facts represented in the names of the chemical elements and their symbols. Listing the names in one column and the elements in another, one can cover one column or the other and proceed from top to bottom in a self-drill. A computer program which merely duplicated this activity would be enormously inefficient in terms of cost/benefit considerations.

The computer could improve on the above drill method by selecting members of the list at random, but this would not significantly increase the cost/benefit factor since such randomness could be practically duplicated by using cards instead of columns for listing the data, and shuffling the cards.

The cost/benefit factor for the computer becomes favorable if, while still selecting data sets at random, the computer were programmed to have a tendancy to select most often those data sets which the user was demonstrably less familiar with and hence needed most drill on. Those data sets which the user matched correctly would be removed from the data bank, but those answered incorrectly would be duplicated, triplicated, etc. (depending on number of wrong responses) thereby increasing the chances of selecting randomly a data set which the user had missed previously. This type of biased random access drill (BRAD) could be done manually by a student and a tutor, but very inefficiently compared with the computer, as the time lag in manually revising the data bank for each student would make the manual project impractical.

ELEMENTS is a BRAD APL program for drilling the element names and symbols. This program features a highly personalized approach to drill methods. Even if many users start with the same body of data, the BRAD feature allows rapid changes in the individual data bases to allow for most frequent drilling of an individual on the sets he is least familiar with. The program also has facilities for user selection and modification of the data base (i.e. a selection of which elements will be tested), and an extensive computer vocabulary for conversational interaction with the user, free of boredom - producing repetition. Details of the operation of ELEMENTS have been described elsewhere (3), the purpose of this brief outline being to show that given that the BRAD technique maximizes communication of drill type information, the computer is the most efficient medium for utilization of this technique.

CHEMBAL

CHEMBAL is a relatively simple program for balancing chemical equations. The equation to be balanced is entered in standard chemical notation, and is immediately printed out again, balanced, with the correct least common coefficients in front of each compound.

One of the best features of IBM 2741 terminals is the ability to use type styles other than standard APL where others would provide more suitable output for the real-life CAIR requirement. Since "standard" chemical notation varies slightly for the type style selected by the user, CHEMBAL includes a vocabulary for the three IBM Selectric styles we commonly use in our CAI programs (APL, Courier, and Chemical Pica) plus provision for programming a fourth style should the need arise. Examples of equations balanced in the three type styles follow:

APL	$3\underline{C}A\underline{O} + \underline{P}2\underline{O}5 \rightarrow \underline{C}A3(\underline{P}\underline{O}4)2$
Courier	**3CaO + P2O5 = Ca3(PO4)2**
Chemical Pica	$3CaO + P_2O_5 = Ca_3(PO_4)_2$

Programs to balance chemical equations are not new, a recent example being a FORTRAN program described by Luteri and Scalzi (4) which, based on the input data, sets up a matrix corresponding to a number of linear simultaneous equations, the solutions of which are some fractional multiple of the desired coefficients of the compounds of the chemical equation when the matrix is reduced to its row echelon form by Gaussian elimination. This mathematical technique for balancing chemical equations dates from the 1940's (5).

While an interesting application of the math technique, the FORTRAN program is not especially suitable, from a pedagogical viewpoint, for chemistry students. First, the output of the program is not a balanced equation, but a list of the compounds with coefficients usually not in least common form. This objection might be corrected with an additional Fortran subroutine. Secondly, the input required is a list of the elements by compound, certainly not any form of standard chemical notation.

For students trying to acquire practice with chemical notation, besides learning techniques of balancing chemical equations, this type of entry is disconcerting and poor teaching technique. However, to enter the data otherwise in FORTRAN would be unfeasible.

APL, on the other hand, readily allows data entry in standard chemical notation, indeed with even a choice of type styles. CHEMBAL then generates the required matrix, which does not have to be reduced to its row echelon form, and uses the domino, ⌹, to produce the least common coefficients in one trival step:

```
[ ] F←F÷⌊/F←⌹Q
```

A few additional statements form the balanced chemical equation output.

CHEMBAL can be used to expand the communicabality of existing CAI programs in chemistry to obviate the need for a library of chemical equations and their balanced forms, or to supplement an existing library to allow students to enter equations of interest to them not otherwise available in the program, or to allow external data to be used with an extant CAI function. The use of CHEMBAL to replace equation libraries is particularly important in systems with workspace size restrictions which limit data storage. It can also be used as the basis for developing new CAI programs which incidentally require the balancing of chemical equations either by the user or the computer. In all of these applications, CHEMBAL allows increased communication flexibility for the programmer and the user while maintaing a real-life input-output syntax essential for maximum computer-user interactivity.

REMOTE CALCOMP PLOTTING VIA APL

In spite of our unabashed enthusiasm for APL for interactive programs, there are situations in which FORTRAN would be the language of choice. One of these circumstances occurs when calculations must be done requiring considerable iteration to reach a result or an approximation of a result. In such cases, APL can consume undesirably large amounts of CPU time, whereas FORTRAN operates quite economically. Because of the economy of RORTRAN and the need for iterations and approximations in plot programs, access to the CALCOMP plotter is restricted to full FORTRAN. Even though a general purpose CALCOMP plotting program in FORTRAN may require a deck as short as 20 cards (excluding data cards), the user still has to punch all the necessary data cards and submit the job at the computing center. Due to this, the CALCOMP plotter at Syracuse University has been under-utilized even though most people who have seen or heard of this valuable tool admire and appreciate its flexibility and the excellent resolution of the drawings, especially when compared to existing APL plotters.

Since the implementation of IJR (Internal Job Reader) at Syracuse, it is possible for APL users to submit any kind of batch job from any teletype terminal on campus using the APL language and all its facilities. We have written an APL function called PLOT which composes a full FORTRAN job, properly formats data card images to fit the user's specifications, submits the deck through IJR so that it will be processed like any other batch job. The advantage of this system is that it is now possible to use the CALCOMP plotter to plot any set of data already stored as an APL variable or generated by any APL function via an interactive APL program which assumes no knowledge on the part of the user of APL or FORTRAN. Thus, PLOT effectively increases the communicability of all who need to communicate ideas via complex plots.

A sample interaction using the PLOT program follows:

SAMPLE INTERACTION USING THE PLOT FUNCTION

```
      PLOT

DO YOU WISH TO RESPECIFY JCL PARAMETERS?  YES OR NO
⎕:
      YES
YOUR NAME IS?                ESKINAZI
THE VALID PASS WORD FOR RUNNING BATCH ON ACCOUNT 14700000 IS?  DEPT__
ESTIMATED EQUIVALENT TIME?  (MINIMUM VALUE IS 1 MINUTE)
⎕:
      2
THOUSANDS OF LINES TO BE PRINTED?  (MINIMUM=1 THOUSAND)
      1
ESTIMATED PLOTTER TIME IN MINUTES?  (MINIMUM=10 MIN.)
⎕:
      12
REGION ESTIMATE IN THOUSANDS OF BYTES?  (MINIMUM= 150K)
⎕:
      150

THANK YOU.  ALL THE REQUIRED PARAMETERS HAVE BEEN RESET.


WHAT IS THE PLOT'S TITLE?
NEW DATA PLOT NUMBER 3

IS THE X AXIS LINEAR OR LOGARITHMIC?
LINEAR
HOW LONG SHOULD THE X AXIS BE?
⎕:
      15

WHAT IS THE X AXIS LABEL?
OBSERVED DATA

IS THE Y AXIS LINEAR OR LOGARITHMIC?
LINEAR
HOW LONG SHOULD THE Y AXIS BE?
⎕:
      10

WHAT IS THE Y AXIS LABEL?
CALCULATED DATA


HOW MANY SIMULTANEOUS PLOTS ARE TO BE RUN ON THE SAME SET OF AXIS?
⎕:
      1

PLEASE ENTER THE FOLLOWING FOR PLOT NUMBER 1
```

```
X VALUES?
⎕:
      X
CORRESPONDING Y VALUES?
⎕:
      DATA 1

WHAT KIND OF CONNECTING LINE?  STRAIGHT, DASHED OR NONE?
⎕:
      STRAIGHT
SHOULD A SYMBOL BE DRAWN AT EACH DATA POINT?  YES OR NO?
⎕:
      NO

*******LAST CALL****** CHECK YOUR DATA.  SHOULD THE PLOT BE SUBMITTED?
⎕:
      YES


JOB NUMBER FOR NEW DATA PLOT NUMBER 3 IS : 65009


THE APL CPU COUNT FOR EXECUTION OF THIS FUNCTION IS 0 MIN 1 SEC AND
      8 SIXTIETHS.
```

CONCLUSIONS

We have used APL examples drawn from our own needs in chemistry to illustrate techniques for and a philosophy of program design for communication with APL with transcend the specifics of program content. The program techniques for drilling chemical elements also can drill Russian verbs. The same matrix techniques used in balancing chemical equations can be used for teaching quantum mechanics. Our CALCOMP PLOT program will handle economic trends as well as electrochemical data. The real significance of the programs described is not the body of information to be communicated, but their focus on maximally utilizing APL facilities to achieve CAIR in program design.

Further information on these programs may be obtained by consulting the references and/or writing the authors.

REFERENCES

1. Davis, L. N. and D. J. Macero, J. Chem. Educ., 49, 758 (1972)
2. Davis, L. N., J. Eskinazi and D. J. Macero, Proc. 4th International APL Conference, 59 (1972)
3. Davis, L. N., C. E. Coffey and D. J. Macero, J. College Sci. Teaching, 2/2, 31 (1972)
4. Luteri, G. F. and F. V. Scalzi, Chemistry, 45/10, 17, (1972)
5. Porges, A., J. Chem. Ed., 21, 266 (1945)

Design of a Simulator
for Open Use of an APL Computer
in Science Teaching

Giovani Bartoli, Luigi Bartolo, Vincenzo Spadavecchia
IBM Scientific Center, Bari, Italy

Educational simulations which emit realistic data for students to grapple with are common. But APL permits students to deal directly with functions, and thus to expect that a simulator will not merely generate appropriate data, but, when its own definitions are laid bare, will reveal an inner structure consistent with the laws it illustrates.

As part of a pilot unit on the use of APL in teaching science, a family of functions was constructed to simulate simple physical experiments. The simulator was constructed in such a way that

1. Students control the simulator's use.
2. Students are forced to consider problems of apparatus and error, from which real observations are never free.
3. The inner structure of the simulator follows the same structure of general rules that students are challenged to discover, and thus (when these definitions are revealed) confirms the students' analyses.

Science as the Induction of Function Definitions

A major activity of science is the attempt to induce general laws from observations. This requires identification of the relevant attributes (i.e. a choice of variables) and statement of the relation between them (i.e. function definition). APL provides a vehicle for expressing clearly the definitions of proposed functions, including nesting them so that functions at one level of discourse may be defined in terms of definitions developed earlier, or as special cases of more general functions. An APL terminal system provideds a means to analyze observational data and also to examine the properties of the functions proposed to explain them.

The central role of the computer in instruction should be clear expression of concepts. The computer should be a tool of inquiry under students' direct control, rather than a vehicle for the presentation or control of programmed instruction (Berry, Falkoff & Iverson, 1970; Berry, Bartoli, dell'Aquila & Spadavecchia, 1971). A program should be not a black box from which emanate data, results, or instructions, but a glass box whose controlling algorithms are both accessible and comprehensible. The student should have both the right and the ability to understand the definition of any function he works with, and to experiment with it at will.

Organization of the Course of which the Simulator is Part

To illustrate this sort of open use of the computer in teaching, a sample sequence on the study of falling bodies was prepared, at a level suitable for advanced secondary students. That material (described in a separate report) was organized in the following way:

1. The basic task is to observe some simple phenomenon (such as the fall of weights dropped from a height), to attempt to define a function which describes the observed behavior, and to revise that definition so that the specific behavior becomes an instance of a more general law.

2. Students are presumed to have some initial familiarity with APL, and with procedures for establishing the definitions of mathematical functions by analyzing tables, mappings, or graphs of their application. (Such a sequence is developed in Iverson's Algebra, 1972.)

3. The interpretation of observational data differs from the analysis of mathematical functions in two important respects:

   a. The presumed underlying functions cannot be directly observed other than through the mediation of an apparatus and procedure.

   b. The observations are subject to error.

   For these reasons the only data directly observable must be understood as arising from a composition of functions, including both error and the systematic effects of the procedure.

4. The student must therefore seek to understand both error and his apparatus as functions, and, as far as possible to find inverses to those functions in order to separate their effects from the law he wishes to study.

5. The random variations present in observational data render inappropriate the elementary techniques for deducing the definition of a mathematical function. Examination of the errors of random data leads to measures of dispersion, and to the mean as a quasi-inverse of random inverse.

6. Comparison of alternative procedures leads to a functional description of some systematic effects of the apparatus.

7. Once functions are defined which adequately describe the given experiments, their definitions can be restated in more general terms. Thus a model which correctly describes the motion of falling bodies can be restated in more general terms of time, position, and acceleration. To do so involves investigations, both empirical and formal, of slope, and the introduction of the concepts of derivative and its inverse.

Required Characteristics of a Simulator

In such a sequence, APL is the medium of exposition used to represent not only the fundamental functions of physics, but also the functions that represent the systematic effects introduced by the apparatus and the random effects of error. APL is also the tool of data analysis. APL has yet another role when it used to as a source of the data that students are to analyze. Given the overall goal of open use of the computer, the simulator must meet the following requirements:

1. The outer structure of the simulator must be general, reflecting the general model that diverse sets of observations can be represented as a composition of three classes of functions: the laws underlying the phenomenon being considered, the systematic effects introduced by the method of study, and errors of measurement.

2. Experimental data is obtained by specifiying the law being studied and the apparatus employed, together with the desired values of the independent variable.

3. It should be impossible for the student to generate data in the absence of an apparatus function and error functions.

4. The definitions of the functions representing the law under study are stated in as general a form as possible:

   a. So that individual experiments or apparatus consist simply of establishing appropriate values for the parameters of a more general law;

b. So that the inner representation of the simulator is consistent with the most general representation of the laws, which are the ultimate objective of instruction.

General-Purpose Outer Simulator

The basic technique employed to meet these requirements was to represent each apparatus by a function which does not generate data directly, but instead passes its principal argument unchanged while establishing (as global variables) the functions and parameters that affect the final observation. In APL systems having an "execute" primitive, the names of functions or entire expressions can be passed as literal vectors; in systems lacking such a primitive, a similar effect is achieved by a defined function *EXECUTE* which branches to call on the appropriate pre-defined function. In this article, an "execute" function is represented by the symbol ⍎.

The outer function of the simulator is called *SIMULATE*; to use it, a student enters the word *SIMULATE* followed by an expression which names the law or phenomenon under study, the apparatus used to record it, and the settings of the apparatus that are desired. For example, if students are studying the motion of falling bodies by timing the fall of weights from various heights with the aid of a manually operated timer, such an expression might be

```
      SIMULATE 'FALL' MANUALTIMED HEIGHTS
```

Such an expression yields an array of results having the same shape as *HEIGHTS*, in which each element of the result represents an observation with that apparatus from the indicated height.

The definition of *SIMULATE* is both general and simple:

```
     ∇ Z←SIMULATE X
[1]    Z←DETERMINISTIC+ERROR
     ∇
```

The definitions of the two components of *SIMULATE* are:

```
     ∇ Z←DETERMINISTIC                  ∇ Z←ERROR
[1]    Z←(⍎LAW)+⍎SYSTEMATIC        [1]    Z←⍎RANDOM
     ∇                                  ∇
```

The variables *LAW*, *SYSTEMATIC*, and *RANDOM* in turn contain references to the appropriate functions, which ultimately include expressions in *X*, the controlling parameter whose values are chosen by the student.

Functions to Represent Apparatus

Several apparatus functions are pre-defined by the instructor, corresponding to the various procedures or devices the class has available. In the study of falling bodies, the students may either observe the time of fall from various heights using various timers, or may record the displacement after various times by using a stroboscopic camera or similar device. The choice of apparatus determines both the measure and the sources of error. For instance, *MANUALTIMED* represents an apparatus used to time the fall from various heights, in which a student at the ground with a stopwatch calls a signal to an assistant at the window of a building, who thereupon drops a standard spherical weight to the ground. The definition of this function passes unchanged the heights (which become the argument *X* of *SIMULATE*), but in passing establishes the other parameters:

```
    ∇ Z←LAWFN MANUALTIMED X
[1]   LAW←'0 0 0 ',LAWFN,' X'
[2]   SYSTEMATIC←'0.2 + INVSOUND X'
[3]   RANDOM←'0 0.15 DISTR X'
[4]   MEASURE←'T̲I̲M̲E̲'
[5]   MEDIUM←AIR
[6]   Z←X
    ∇
```

The foregoing shows that the initial time, position, and velocity are to be considered 0 (statement 1). The procedure is subject to systematic effects because of the use of a sound signal, and to a systematic delay due to the response time of the experimenters. Thus *SYSTEMATIC* includes a constant .2 seconds plus a component inversely related to the speed of sound. Observations are affected by random errors with mean 0 and standard deviation .15 secs. The measure of interest is time. The medium through which the fall takes place is air.

Functions Representing the Laws Governing Fall

The function controlling the lawful portion of the motion of a falling body stipulates the forces that apply to it and the arguments of the relevant measure (established previously by the apparatus function):

```
    ∇ Z←C FALL X
[1]   FORCE←'(MASS×GRAVITY)-MEDIUM DRAG V'
[2]   Z←⍎'C ',MEASURE,' X'
    ∇
```

The simulation includes explicit consideration of the drag, which, as can been seen, is a function of the velocity of the falling sphere and the medium through which it moves. This permits the simulator to produce reasonable answers for heights considerably greater than students can themselves use in a laboratory, and also permits them subsequently to consider motion in a vacuum or in other fluids.

Since the apparatus function specified that the measure is to be time, the function *TIME* performs the actual calculation.

```
     ∇ Z←C TIME X; T0; S0; V0; ΔX
[1]    T0←C[1]
[2]    S0←C[2]
[3]    V0←C[3]
[4]    ΔX←D S0,GRADEUP SELECT,X
[5]    Z←T0++\V0 ΔTIME ΔX
[6]    Z←X ORDER Z
     ∇
```

The times are calculated as the cumulative sums of a function *ΔTIME* applied to the initial velocity and the difference in position. (The functions *GRADEUP* and *SELECT* arrange the argument as an ordered set without duplicates, while *ORDER* restores the results to match whatever order the student supplied as the argument.)

Definition in Terms of a General Law of Motion

The amount of change in time depends upon the acceleration, which, following Newton's law, is a function of the mass of the body and the forces acting on it:

```
     ∇ Z←NEWTONLAW
[1]    Z←(⍎FORCE)÷MASS
     ∇
```

In the case of fall, the function *FALL* has established that the force is the effect of gravity minus the effect of drag. The function *ΔTIME* then evaluates the double integral *S PRIMITIVE V PRIMITIVE A* in order to find the coefficients of the polynomial for position as a function of time. It then solves for the elapsed time to reach the given position.

```
     ∇ Z←V ΔTIME ΔX; S; A
[1]    →0 IF(ρΔX)=ρZ←⍳0
[2]    S←0
[3]    Z←(1↑ΔX) SOLVE S PRIMITIVE V PRIMITIVE A←NEWTONLAW
[4]    Z←Z,((1↑Z) POL V PRIMITIVE A) ΔTIME 1↓ΔX
     ∇
```

Statement 3 of *ΔTIME* computes the time needed to reach the first recorded position. To this is catenated (statement 4) the time needed to reach the next, based upon the same formula, except that now for the initial velocity is substituted the velocity attained at the preceding position.

(The primitive of a function is the inverse of the derivative; in the case of polynomial functions, the two definitions are:

```
    ∇ Z←DERIVATIVE C                 ∇ C←K PRIMITIVE A
[1]   Z←1↓C×¯1+⍳⍴C              [1]   C←K,A÷⍳⍴,A
    ∇                                ∇
```

The function *POL* evaluates a polynomial whose coefficients are its right argument, while *SOLVE* is its inverse for a given set of coefficients, making use of an iterative function *ROOT*:

```
    ∇ Z←X POL C
[1]   Z←(X∘.×⍳1)⊥⌽C
    ∇

    ∇ Z←X SOLVE C
[1]   Z←1 ROOT((1↑C)-X),1↓C
    ∇

    ∇ Z←X ROOT C
[1]   Z←X
[2]   X←X-(X POL C)÷Z POL DERIVATIVE C
[3]   →1 IF Z≠X
    ∇                                         )
```

A definition analogous to that of *ΔTIME* is supplied for the amount of displacement (required for an apparatus such as a camera which records position rather than time):

```
    ∇ Z←V DISPLACEMENT ΔX; S; A
[1]   →0 IF(⍴ΔX)=⍴Z←⍳0
[2]   S←0
[3]   Z←(1↑ΔX) POL S PRIMITIVE V PRIMITIVE A←NEWTONLAW
[4]   Z←Z,((1↑ΔX) POL V PRIMITIVE A) DISPLACEMENT 1↓ΔX
    ∇
```

Allowance for Drag

Experiments conducted in air are subject to the effects of drag. For small bodies, these have the effect of severely limiting velocity, so that simulated experiments from large heights will produce results quite different from those in a vacuum. The simulator incorporates these, although they may go beyond what beginning students are likely to require. The drag is approximated by either of two functions, *STOKES* or *REYNOLDS*, depending upon the Reynolds number appropriate to the medium:

```
    ∇ Z←MEDIUM DRAG V; DENSITY; VISCOSITY; RE
[1]    →(Z←0) IF MEDIUM=VACUUM
[2]    DENSITY←1↑MEDIUM
[3]    VISCOSITY←¯1↑MEDIUM
[4]    RE←2×RAD×DENSITY×(|V)÷VISCOSITY
[5]    →ST IF RE≤1
[6]    Z←MEDIUM REYNOLDS V
[7]    →0
[8]   ST:Z←MEDIUM STOKES V
    ∇

    ∇ Z←MEDIUM STOKES V; VISCOSITY
[1]    VISCOSITY←¯1↑MEDIUM
[2]    Z←○6×VISCOSITY×RAD×V
    ∇

    ∇ Z←MEDIUM REYNOLDS V; DENSITY; VISCOSITY; RE; AREA; DC
[1]    DENSITY←1↑MEDIUM
[2]    VISCOSITY←¯1↑MEDIUM
[3]    RE←2×RAD×DENSITY÷VISCOSITY÷|V
[4]    AREA←○RAD*2
[5]    DC←(R̲E̲⌊RE) INTERPOLATE RE I̲NTERVAL DRAGTABLE
[6]    Z←0.5×DC×AREA×DENSITY×V*2
    ∇
```

This function requires interpolation in the table of drag coefficients *DRAGTABLE*. This is a 6x6x2 array, but to save space the successive planes are here arranged horizontally across the page:

```
 1 25.0      10 4.0        10 1.10      1000 .45      10000 .40      100000 .40
 2 15.0      20 2.5      200   .80      2000 .40      20000 .45      200000 .40
 4  8.5      40 1.9      400   .60      4000 .40      40000 .45      300000 .20
 6  6.0      60 1.5      600   .55      6000 .40      60000 .45      400000 .09
 8  5.0      80 1.2      800   .50      8000 .40      80000 .45      600000 .10
10  4.0     100 1.1     1000   .45     10000 .40     100000 .40      800000 .12
```

Summary

The use of this type of simulator permits the student to conduct his own experiments, using a wide range of apparatus and conditions. During these early phases he should refrain from displaying the definitions of the functions (or the instructor may lock them). By developing the appropriate questions, the instructor can lead students to an appreciation of the effects of random error, to the possibility of systematic error (and, in this example, to measures of such systematic factors as the speed of sound and human reaction time). Students may then find that, at least for a moderate range of heights, position of a falling body is reasonably approximated by a second degree polynomial.

Having reached that point, the instructor may seek a more general restatement of the initial ad hoc simulator. He may use the computer empirically to examine the slopes of functions, and present a general statement for the slope (i.e. the derivative) of a polynomial. From there he may approach the inverse of that function, so that any particular motion can be described in terms of the forces acting upon it (i.e. its acceleration) and its initial speed and position.

When this development is complete, students may return to the simulator and examine its internal stucture. The general form of the laws that they have been asked to find are clearly manifest in the simulator itself.

References

Berry, P.C., A.D. Falkoff & K.E. Iverson, Using the Computer to Compute: a Direct but Neglected Approach to Teaching Mathematics. IFIP World Conference on Education, Amsterdam, 1970. Also available as Technical Report 320-2988, IBM Corporation, Philadelphia Scientific Center, 1970.

Berry, P.C., G. Bartoli, C. dell'Aquila & V. Spadavecchia. "APL and Insight," Colloque APL, IRIA, Rocquencourt, France, 1971.

Iverson, K.E. Algebra: An Algorithmic Treatment. Palo Alto: Addison-Wesley, 1972.

APL AS AN AID TO THE TEACHING OF UNDERGRADUATE ECONOMICS

J. R. Savary
Assistant Professor of Economics
Glendon College, York University

All of us who have been involved, if only peripherally, in using computers in connection with our academic work must have been impressed with its potential as a teaching aid. Papers given at this and other conferences reflect this enthusiasm, and indeed testify to the growing use of APL as a language very well adapted to CAI. The impression one gets from reading these papers, however, is that the use of APL as an aid to education is restricted to the more formal question-answer approach where the interactive capabilities of the language render it a very powerful teaching tool. Moreover, the applications reported on have generally been in the physical sciences, mathematics, or business.[1] What seems to have been generally overlooked is that APL is very well suited to allowing students at an early stage in their university education to use a computer for what it is really intended to do, i. e., compute. The purpose of this paper is to discuss my experience in using APL with my third and fourth year undergraduate students in Economics at Glendon College.

---

[1] Leslie N. Davis, Jak Eskinazi, and Daniel J. Macero,"APL as a Teaching Tool: Two Versatile Tutorial Approaches," Proceedings of the Fourth International APL Users' Conference,(Atlanta, Georgia: June 15-16, 1972); Paul Penfield, Jr., "Use of APL in Teaching Electrical Network Theory", Proceedings of the Fourth International APL Users' Conference ,(Atlanta, Georgia: June 15-16, 1972); P. Meinke, "Computer Assisted Learning in APL", Proceedings, York University APL Conference, (York University, Toronto, Ontario: January 27, 1972); R. Roden, "The Use of APL in Teaching Mathematics at the University of Waterloo," Proceedings, York University APL Conference,(York University, Toronto, Ontario: January 27, 1972).

The social sciences generally are unable to test hypotheses in the controlled experiment permitted in the physical sciences. Therefore, much emphasis must perforce be placed upon the analysis of large amounts of data through such statistical techniques as multiple regression. Economics often claims to be the most empirically oriented of the social sciences. Yet the teaching of Economics at the undergraduate level seldom reflects this in any meaningful way, except in the purely mathematical or statistical courses. With the advent of computer timesharing using APL, our experience suggests that students welcome greater exposure to quantitative methods when they are not required to punch cards and learn difficult programming languages.

There seem to me to be three features of APL, which, in combination, make it particularly suitable as a language for introducing undergraduate students to data analysis. First, the APL system is a timesharing one, accepting input and producing output at remote terminals in several locations. This makes the system highly advantageous to small institutions unable to afford their own computer facilities, or institutions like Glendon College, an integral part of a large university but with its own campus located some distance away from the main campus of the university. Other languages also permit the use of remote terminals, but the hardware required may be prohibitively expensive for small institutions.

The second advantage of APL is its fully interactive nature. From the point of view of teaching undergraduates, this feature seems to me to be of paramount importance. It permits the writing of programmes which allow the student to access and use the computer without knowing any computer language at all. Thus it is not necessary to spend class time trying to give a "quickie" course in programming, key punching, etc., the student need know only how to sign on, load a function, enter his data, and sign off. These procedures can, of course, be easily taught and demonstrated in less than one class hour. In a few moments, I will briefly outline as an example the general programme for time series analysis which I have compiled for student use.

The third very useful feature of APL as a teaching and learning aid is the ease with which the language itself can be learned. Although there is no doubt that much practice would be required before the user is able to employ the full power and elegance of the language, a fair degree of competence can be achieved simply by working through one of the texts on APL.[2] This implies that instructors can write programmes specifically suited to the material they are covering, or modify existing programmes in the public libraries. Moreover, my experience has been that students, once exposed to APL, often display an interest in learning the language, which has in turn increased their interest in quantitative analysis. Finally, the ease of learning the language permits one to economize on staff; it is not necessary for each institution to employ several operators and programmers in order to use the system.[3] This again may be of considerable importance to smaller institutions, and may be a vital factor in any decision concerning whether to permit undergraduates to use the computer.

Let me turn now to an example of an actual application of APL.

---

[2] An excellent general APL text, well suited for self-teaching, is Leonard Gilman and Allen J. Rose, APL/360: An Interactive Approach (John Wiley & Sons, Inc., New York: 1970). On York APL, see J. Morgan Smyth, York APL (Ryerson Polytechnical Institute, Toronto: 1972)

[3] This is not to denigrate the value of professional programmers. York University's Computer Services has been of immense help to myself and my students whenever problems arose in the writing and execution of programmes. I only wish to point out that with APL one need not have several operators and programmers at the remote terminal.

As I pointed out earlier, quantitative economics quite often involves the testing of hypotheses through the analysis of time series. In order to permit my upper year students to follow standard analytical techniques, I designed an interactive programme which is built around a multiple regression function. A master programme is employed, using the unquote feature of YORK APL to overlay extensively. Sub functions consist of modified versions of the simple and partial correlation functions in the York public library,[4] and my own functions for data transformation, multiple regression, the generation of Almon distributed lags,[5] and an autoregressive transformation[6] to remove possible serial correlation. The programme is designed to permit considerable flexibility. It works as follows:

1) The student signs on and loads the programme ANALYZE.

2) He then enters his data in matrix format, using the rho operator. It would be easy to design the programme to accept data, one row at a time, but in practice my students have preferred to enter and save their data separately. If the data has been previously saved, the student will of course load it from his library.

3) The student then types ANALYZE X, where X is, as usual, the name of his matrix. The computer responds by asking if he wishes a data transformation. If he types YES, the computer asks him to specify the details of the desired transformation. The information is submitted to the computer in the form of a four element vector. The first element

---

[4] These in turn are modifications of programmes written by K. Smillie of the University of Alberta.

[5] The initial version of this function was written in the spring of 1972 by M. Charbonneau, then an Economics student at Glendon College and now at the University of Western Ontario.

[6] The initial version of this function was written by P. Dungan, in 1972, then an Economics student at Glendon College and now at Princeton University.

designates the column to be transformed and the second the weight given to that column. The third element (1 to 6 inclusive) designates the transformation desired (1-reciprocal, 2-log, 3-squared, 4-ratio, 5-percentage change, 6- first differences). The fourth element (0 or 1) permits the student to choose simple lag patterns. The function will loop to permit various combinations until END is typed.

4) The computer then asks whether correlation coefficients are desired. If the student types yes, a matrix of simple correlation coefficients is printed. He is then asked if he wishes to have partial correlation coefficients. If he types yes, a matrix of partial correlation coefficients is printed.

5) The student is then asked whether he wishes to use Almon lags. If he answers yes, the data is transformed and weighted according to Almon's procedure; the student is then asked to specify the column to be lagged, the degree of the polynomial along which the weights are to lie, and the number of periods to be lagged. He is then asked to specify the column numbers of any non-lagged independent variable, and the dependent variable.

6) If the student has answered no when asked whether he wishes Almon lags, the regression is then performed on the data as it stands. (Note that it may have been transformed at an earlier stage). Whether Almon lags are desired or not, the input to and output from the rest of the programme is similar. The student is asked whether he wishes the origin constrained to be zero, in response to which he types YES or NO. The computer then prints out the results of the regression in the form shown in TABLE I.

Residuals may be printed as an option, and tests on residuals provided·Upon examination of the statistical parameters, primarily the Durbin-Watson statistic , the student may wish to have an autoregressive transformation. Upon a positive response to the computer's query, the data is transformed, a new regression performed, and the new output printed. The computer then asks whether more regressions are required

TABLE I

```
TITLE


DEPENDENT VARIABLE         1
INDEPENDENT VARIABLES        2  3

          VAR.     COEFF.      ST. ERROR    T-VALUE
           2      .19138        .03499       5.469
           3      1.6114     14.36159         .112

INTERCEPT                      ¯93.96733139
INTERCEPT STD. ERROR            28.28845
INTERCEPT T STATISTIC          ¯3.322

REGRESSION
   DEGREES OF FREEDOM           2
   SUM OF SQUARES               105026.4052
   MEAN SQUARE                  52513.20262

ERROR
   DEGREES OF FREEDOM           20
   SUM OF SQUARES               14502.12233
   MEAN SQUARE                  725.10612

S.E. ERROR OF ESTIMATE          26.92779
F-VALUE                         72.42
MULTIPLE R-SQUARED              87.87
ADJUSTED MULTIPLE R-SQUARED     86.65
```

with the same data. A positive response causes a loop back to the point where the student is asked whether he wishes to use Almon lags.

The initial version of the programme was first used during the academic year 1971-72 and made available to students in Monetary Analysis. It consisted solely of linking several functions in the York public library, plus the Almon lag functions. Neither a transformation function nor an autoregression function were available, and the correlation functions and the Almon lag function had to be loaded and executed separately. There was no provision for constraining the intercept to zero, nor were significant statistics produced for the intercept. Moreover, the programme was only partly conversational, requiring the students to have some knowledge of APL. Yet several students used the programme in undertaking quantitative papers in their courses in Monetary Analysis, International Trade, Regional Economics, and Econometrics. Moreover, two of the students became well enough versed in APL to participate materially in the improvement of the programme. The revised version, available only since Christmas, is being used by more students and is superior in that it is fully conversational and permits a higher level of statistical analysis.

It is difficult to assess the effect on student performance of the availability of APL. The use of this system is completely voluntary. Students making use of it have done excellent work, but to some extent the process is a self-selecting one; it is the best students who have selected difficult essay topics that use the system. I can safely say, however, that those students which have used the computer have received an introduction to empirical research which they otherwise would not ordinarily receive until their later postgraduate years. Reaction from last years students presently employed or at graduate school has been that they have found their exposure to computer-assisted empirical research to be an immense advantage.

The success of these initial attempts suggests that extensions would be worthwhile. Two approaches suggest themselves. The first is to develop methods of assessing government data tapes through a file system, which would permit easy access to large quantities of detailed information. With the assistance of Mr. D. Kellett of York University's

Computer Services, two of my students are presently working on projects which do make use of the Statistics Canada CANSIM tape. Although the relatively small workspace size under York APL does provide a constraint to the amount of data which can be conveniently handled, Mr. Kellett has suggested ways of at least partially overcoming this.

The second extension which suggests itself is to begin student exposure to APL at a much earlier stage in the student's university career. Several papers on CAI will be given during our meetings, so I only want to note here that Economics at the introductory level seems particularly suitable to CAI. A glance at any of the student workbooks accompanying the standard principles texts indicates the ease with which the problem sets could be treated so that student self-teaching could be achieved, Moreover, the calculation of consumption functions and the various Keynesian style multipliers could be made much more realistic, and students might even be encouraged to try their hands at (crude) forecasting. The programming would, of course, need to be fully conversational, and might follow the lines of that discussed by Professors Davis, Eskinazi and Macero at APL IV.[7]

It would be remiss not to also note the possible problems involved in encouraging undergraduate students to use the computer in doing assignments. The foremost difficulty is the fact that students not fully conversant with the principles of statistical inference may use inappropriate techniques. Unless there is reason to expect the assumptions underlying ordinary least squares to hold, the use of that procedure may lead to erroneous conclusions. So far that has not been a problem at Glendon, since students using the system have all had courses in statistics and econometrics. It is likely, however, that the ease with which time series analysis can be carried out may lead to it becoming a problem in the future, and close supervision of student work at the formative stage will be essential.

---

[7] Davis, Eskinazi and Macero, "APL as a Teaching Tool," op. cit.

The second difficulty is the matter of cost, and hence terminal availability. An unfortunate effect of the budget stringency faced by many universities has been the pressure brought on administrations to reduce "non-teaching" (i. e., non-faculty) expenditures. This may make it difficult for the university to finance the large, and growing, number of terminals that will be required if APL is to be made readily available to undergraduate students. Only if administrations and faculty associations can be convinced that comprehensive computer facilities are an integral part of teaching as well as research is it likely that meanginful extension of services to undergraduates can be achieved.

Let me conclude on a more optimistic note by summarizing what I think are the main results of our experience with APL at Glendon College. First, APL is uniquely suited for use by undergraduate students because it is interactive, permits the use of remote terminals, and is relatively easy to learn. Second, the introduction of undergraduate students to computer-assisted empirical analysis can lead to work of excellent calibre. Finally, students are likely to find their undergraduate preparation much more useful in their subsequent work, whether this be in employment or in graduate studies. These factors suggest that it will be well worthwhile to extend our use of the computer in the teaching of Economics, and indeed in the Social Sciences generally.

April, 1973

INVITED PAPER

APL AS AN ANALYTIC NOTATION

BY

K. E. IVERSON

APL is much more than a computer programming language. It is a concise and convenient notation which can greatly aid the process of analytic thought.

Formal Differentiation Using APL

By

G. Wilhelmi

1. Introduction

In this paper the use of APL for formal differentiation is discussed as an example of problems in nonnumerical mathematics and symbol manipulation. In implementing APL-functions to perform formal differentiation of APL-expressions I pursued 3 aims :

a. to investigate how effectively APL can be used in problems of symbolic mathematical computation;

b. to find out how expressions can be simplified along with differentiation;

c. to construct an APL-function which can be a useful tool for numerical problems as soon as an unquote operator is available.

To avoid mixing up of functions to be differentiated and APL functions, it should be clearly noted that the following paragraphs will deal with the formal differentiation of APL-expressions. Two workspaces of APL functions have been written. One workspace called CHECK comprises about 15K of APL functions for the syntax check of APL expressions contained in characterstrings, for the removal of superfluous parenthesis, as well as functions for plotting the tree structure of the APL expression. We will mainly be concerned with the second workspace which contains in about 35K the function D and the ancillary functions actually performing the formal differentiation. The function D has the following syntax :

DERIVATIVE ← DIFVAR D STRING

Here the parameters DIFVAR and STRING and the resulting DERIVATIVE are characterstrings . DIFVAR contains the name of the differentiation variable, STRING contains any legal APL expression using the monadic and dyadic operators + - × ÷ * ⍟ ○ , parentheses and legal APL constants and variable names (No blanks and superfluous parentheses are permitted; they may be removed in the workspace CHECK). The resulting characterstring DERIVATIVE contains the formal derivative of STRING and represents a legal APL expression which may be executed if an unquote operator is available.

2. Simplification of Expressions

A problem closely connected with formal differentiation but by far more difficult to solve is the simplification of expressions. Any useful program which performs formal differentiation must be directed towards a simplification of expressions . This problem can be handled by either performing simplification along with differentiation or after differentiation. I decided to do some simplification during the differentiation process and no simplification afterwards. The kind of simplification which can be easily performed highly depends on the data representation chosen for the expression. I will comment on this problem in greater detail below. The advantage of my approach to this problem is the fact that one does not get involved in the general problem of complicated symbol manipulation and in general still achieves tolerable result expressions (compare Section 4. Results). There is of course the disadvantage that, depending on the expression, a lot of simplification may still be possible.

If a partial expression is to be differentiated, the leading operator of this expression is looked for at every point in the differentiation process. Every dyadic partial expression can now be represented by A ⊙ B, where A and B themselves are partial expressions and ⊙ is any APL operator. The shortest possible form of the derivative of this partial expression is chosen according to the fact whether A and B contain the differentiation variable, do not contain the differentiation variable, or are identical to the differentiation variable. There may be up to 9 different cases to be considered. Let us consider as an example the operator x and few of these cases (differentiation variable X) :

$$(A\times B)' = \begin{cases} A'\times B + A\times B' & \text{in general} \\ A'\times B + A & \text{if B does not contain X} \\ 2\times X & \text{if A equals X and B equals X} \\ B & \text{if B does not contain X and A equals X} \\ 0 & \text{if A and B do not contain X} \end{cases}$$

It should be clearly noted that in this simplification process the need of numerical computation arises at one point. The result of 'X' D 'X*3' should be 3xX*2 rather than 3xX*3-1. Here again the benefit of an unquote operator is evident.

and running through all operators of the tree, the differentiation process builds up the tree structure of the derivative. This view of formal differntiation has a strong geometical appeal and can be easily implemented in APL. Futhermore, the use of subtrees is of considerable help in the simplification of the APL expressions during the differentiation ; because it can be easily decided whether a subtree contains or does not contain the differentiation variable.

As there is no direct way to describe a tree structure in APL/360, I decided to describe the tree in a matrix of two rows. It is found that the order of operands and operators in the APL expression is the same as the order of the tree nodes projected on the abscissa of a coordinate system in the plane. So one coordinate of a node is equivalent to its position in the APL expression. The second coordinate corresponds to the level of the node in the tree. Let us arbitrarely assign the level number 99 to the highest level in the tree. Furthermore, if we represent variables and operators by different integers, for example according to the following table,

3	+
4	-
5	×
6	÷
7	*
8	⍟
9	○
11	*CONSTANT OR VARIABLE*

the matrix corresponding to the tree structure in our example would be

$$\begin{matrix} 11 & 5 & 8 & 11 & 3 & 11 & 4 & 11 & 6 & 11 & 7 & 11 \\ 97 & 98 & 97 & 95 & 96 & 95 & 99 & 97 & 98 & 96 & 97 & 96 \end{matrix}$$

This matrix representation directly reflects the coordinates of the nodes of the tree, but does not indicate the branches of the tree . But it can be easily proved that the branches of this binary tree contain no information , if the coordinates are known. They can be reconstructed from the matrix.

So the first step is to map an APL expression onto the matrix corresponding to the tree structure of the expression. In this mapping information contained in the parentheses and the relative position of the operators is transformed into information about the tree level of the operator. This can be accomplished using a fairly simple algorithm. As indicated above , this one-to-one mapping and the APL expression can be reconstructed from the matrix. In order to manipulate subtrees and for the purpose of simplification the matrix as described above is enlarged by further rows describing the extent of the right and left subtrees belonging to each operator. More information about the tree is contained in some global variables (e. g. , the existence of the differentiation variable in every subtree).

The differentiation process starts at the operator of the highest tree level and the two subtrees belonging to this operator. This can be considered as an elementary expression of the form

(*) $A \bullet B$

where A and B are subtrees described by matrices. In the workspace we have a collection of APL functions which produce for the monadic and dyadic APL operators + - x ÷ * ●o a matrix which constitutes the derivative of this expression. This is done with regard to possible simplifications as described above. Substituting this matrix in (*) results in a new tree consisting of operators, numbers, the differentiation variable, subtrees of the original tree, and subtrees of the original tree to be differentiated. The next step is now to choose any subtree still to be differentiated. This subtree is again of the form (*). So , as described above, the differentiation of this subtree simply means the insertion of a new subtree replacing a node. This can be effectively done in APL using simple matrix operations.

This process is continued until, finally, there is no subtree left which has still to be differentiated. No recursive functions are used, as the process can be controlled by a simple iteration. It is noteworthy that only those subtrees containing the differentiation variable must be explicately differentiated, which may result in an important simplification of the derivative.

As a final step the resulting matrix must be retransformed into the APL characterstring . This results in the inverse process of inserting parentheses in the projection of the resulting tree structure on the abscissa according to the level of the operators.

3. Formal Differentiation and Data Representation.

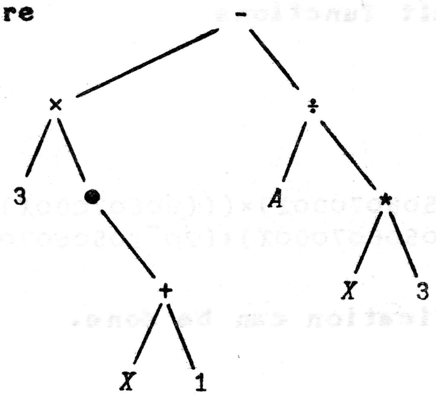
One of the main points to consider is the choice of the internal representation of the APL expressions. The APL characterstring denoting the expression can be thought of as a linear representation of APL expressions. Polish notation (prefix and postfix) are different ways to represent the same expression in a linear string without using parentheses. Another well-known two-dimensional way to represent an expression is a tree structure. For example the APL expression

(3 × ● X + 1) − A ÷ X * 3

may be equivalently represented in prefix polish notation

− × 3 ● + X 1 ÷ A * X 3

(here one would need a separation symbol to be unambiguous) or as a tree structure



In the tree structure representation the process of formal differntiation can be easily described in terms of manipulation of partial trees. According to the type of operator considered, starting with the 'highest level' operator (Minus in our example)

4. Examples

I would like to demonstrate by some examples the amount of simplification which can be obtained when using this method.

a. Polynomials

Using the traditional representation

`(2xX*3 ) + ( 3xX*2 ) + (4xX ) + 5`

results in the derivative

`( 2x3xX*2) + ( 3x2xX*1 ) + 4`

while the more APL-like form

`5 + Xx4 + Xx3 + Xx2`

results in a more complicated form

`(Xx(Xx2 )+3+ Xx2 )+4+Xx3+Xx2`

b. Differentiaition of implicit functions

```
 'X' D '1○2○3○¯1○5○6○7○0○X'
```

results in

```
(2○2○3○¯1○5○6○7○0○X)×(-1○3○¯1○5○6○7○0○X)×(((6○6○7○0○X)×(5○7○0○
      X)×(-X÷0○X)÷(6○0○X)*2)÷0○5○6○7○0○X)÷(2○¯1○5○6○7○0○X)*2
```

Here not much further simplification can be done.

c. Two Examples of Multiple Differentiation

```
 'X'D'X'D'X'D 'X×*X'

      (((X×*X)+*X)+*X)+*X

'Y'D'X'D '(1○X)×X*3+Y*3'

      (2○X)×3×Y*2
```

5. Conclusions

There are several possibilities of extending the work I have done so far. One can for example allow in the APL expression any functions written by the user. Of course, this gives rise to new problems as in this case there is no longer a closed system of operators and functions. A new way must be provided to generate function names for derivatives. In case of inclusion of dyadic and monadic functions there is in general no way to distinguish between them. One should also consider to do some kind of simplification after differentiation as proposed by example a. In these problems the unquote operator will be of some importance I decided to postpone this work until it will be available in APL/360.

PROPOSED NOTATION AND IMPLEMENTATION FOR DERIVATIVES IN APL*

Paul Penfield Jr.**

Abstract: The lack of notation for derivatives in APL is an important barrier to its acceptance in many disciplines. This paper describes a notation for derivatives and a procedure for evaluating expressions containing derivatives.

Operators and Functions

In describing APL syntax, it is frequently useful to distinguish functions from operators. Primitive scalar functions, primitive mixed functions, and defined functions are all "functions." The term "operator" is used for a symbol which acts on one or more functions or variables, and produces, as a result, a new function. The new function then acts on its arguments and produces its result, an APL array. The operator, of course, must act before the function can.

There are three operators implemented in APL: reduction, inner product, and outer product. There has been suggested also a "scan" operator. These operators can, in principle, operate on defined functions as well as primitive functions, although they are not now so implemented. The derivative operator, discussed below, can likewise be used on both primitive and defined functions, but the implementation would probably be of little benefit if it were restricted to primitive functions.

Expectations of this Notation

There are three reasonable expectations for a notation for derivatives. First, we might expect the notation to denote (and the implementation to calculate) the result of applying the new function produced by the operator, to an argument; that is, to denote the returned array. Second, if the purpose of such an operator is to return another function, one might wish the notation to denote (and the implementation to display) the resultant function. Third, we might expect numerical differentiation of functions defined only by numerical values.

Of these three expectations, the second is difficult, both conceptually and practically, and the third is prone to numerical inaccuracy. The notation and implementation given in this paper fulfill the first expectation but not the others.

One might also expect that it should be possible to devise a notation for integrals. The integral case seems to be much more difficult, however.

---

* This work was supported in part by the National Aeronautics and Space Administration, NASA Grant NGL 22-009-337.

** Department of Electrical Engineering and Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Types of Derivatives

It may not be out of place to recall the wide variety of derivatives that are encountered in mathematically based disciplines. These include the ordinary total derivative, partial derivatives, the substantive derivative of fluid mechanics, and the gradient, divergence, and curl in field theory, as well as high-order derivatives such as scalar and vector Laplacians. Derivatives are taken with respect to any argument of a function, or with respect to any parameter.

The proposed notation not only covers all these cases, but can be further generalized. Differentiation is allowed not only of functions with respect to arguments, but also of functions or expressions with respect to global variables.

Graphics

Two new symbols are required for the derivative notation, one for an operator and the other for a function. The symbols ♭ and ⊧ are used here, without any implication that they are optimum.

Notation for Derivatives of Monadic Functions

Consider first a monadic primitive scalar function, with a scalar argument. The syntax for this function is *Z←F A*; the derivative of *F* with respect to *A*, evaluated at the particular value of *A*, is denoted *F♭A*. For example, +♭3 is equal to 1; ÷♭2 is equal to ¯0.25; and ⌊♭3.5 is equal to 0.

Next, consider any monadic function, such as a primitive scalar function, a primitive mixed function, or a defined function. Again, the syntax is *Z←F A*. The derivative of *F* with respect to *A* is denoted *F♭A*, and is defined provided *A* and *F A* are conformable. The rule for conformability is the same as the rule for the expression *A+F A*; that is, either *A* and *F A* have the same dimensions, or else one of them has only one element. If *A* is a scalar and *F A* is an array, then *F♭A* consists of the derivative of each element of *F A* with respect to *A*, evaluated, of course, at the value of *A* given. For example, if the velocity *V* of a particle and its position *R* are both functions of time *T*, and velocity and position in 3-space are represented by vectors of length 3, then the velocity is the derivative of the position, as expressed by the equation (*V T*)=*R♭T*.

If *A* and *F A* have the same dimension, then *F♭A* consists of the partial derivative of each element of *F A* with respect to the corresponding element of *A*. In general, each element of the results depends on the values of all the elements in *A*, and is, of course, evaluated for the *A* in question. This form of the derivative is useful, for example, in describing waveforms as a function of time: if *F T* is the function 2×*¯5×*T* which might be the natural response of a first-order physical system, then *F♭T* would calculate an array consisting of the derivative of *F* evaluated at each of the times in the array *T*. An example from economics: the price elasticity of demand, where the quantity demanded *Q* is a function of the price *P*, is (*Q♭P*)×*P*÷*Q P*.

Finally, if *F* is a scalar function of a vector argument *R* which might stand for position in space, then *F♭R* is a vector, each element of which is the partial derivative of *F* with respect to the corresponding element of *R*; that is *F♭R* is the gradient of *F*. Thus, the relationship between electric-field vector *E* and potential *PHI* in electrostatics can be written in the form *E*=-*PHI♭R*. As another example, (×/♭*A*)=(×/*A*)÷*A* (except that it works properly even if 0∊*A*). In the last example, note that the reduction operator acts before the differentiation operator.

Outer Derivatives

It is frequently necessary to denote the partial derivative of each element of the result of a function, with respect to each element of its argument. This concept resembles that of the outer product, and a similar notation is suggested. Thus, for a monadic function *F*, *F*∘.ƀ*A* is called the "outer derivative". There are no conformability requirements; the outer derivative has dimension (⍴*F*∘.ƀ*A*)=(⍴*F A*),⍴*A*. An example of the outer derivative is the velocity-gradient tensor in fluid mechanics, of which the rotation is the antisymmetric part and the strain rate is the symmetric part. Thus, if the velocity *V* is a function of space *R*, then the rotation is 0.5×(*V*∘.ƀ*R*)-⍉*V*∘.ƀ*R* and the strain-rate tensor is 0.5×(*V*∘.ƀ*R*)+⍉*V*∘.ƀ*R*. As another example, +∘.ƀ⍳*N* is the unit matrix of dimension *N*,*N*.

Inner Derivatives of Monadic Functions

The inner derivative of a function *F* is denoted *F*+.ƀ*A*, and the conformability requirements are the same as for the inner product (*F A*)+.×*A*. As an example, one of Maxwell's equations in electromagnetic field theory states that the divergence of the magnetic field *B* is zero, or 0=*B*+.ƀ*R*.

Generalizations of the inner derivative are possible by using scalar dyadic functions other than +. For example, *F*×.ƀ*A*. For a function *F T* of time, its largest derivative at any time in a vector *T* is given by *F*⌈.ƀ*T*.

High-Order Derivatives of Monadic Functions

Second and higher-order derivatives of monadic functions are denoted similarly, with multiple use of the derivative operator, for example *F*ƀƀ*T*. Thus, *F*∘.ƀ∘.ƀ*A* is the collection of all possible second-order partial derivatives of *F*, and has dimension equal to (⍴*F A*),(⍴*A*),⍴*A*.

As an example, consider Newton's law for a particle, where *F* is the force, *M* is the mass, and *A* is the acceleration, the second derivative of its position *R* with respect to the vector of times *T*: *F*=*M*×*R*ƀƀ*T*. As another example, the homogeneous differential equation obeyed by a series RLC electrical network: 0=(*L*×*I*ƀƀ*T*)+(*R*×*I*ƀ*T*)+(*I T*)÷*C*. Another interesting second-order example is the Laplacian of a scalar function *F* of space coordinates *R*, which is *F*∘.ƀ+.ƀ*R*. Note that the outer derivative is performed before the inner derivative; that is, the derivative on the left before the derivative on the right, in contrast to the normal APL convention about order of execution of functions. If the function *V R* is a vector function of *R*, then the vector Laplacian is *V*∘.ƀ+.ƀ*R*.

Derivatives of Dyadic Functions

The notation for derivatives, inner derivatives, and outer derivatives of dyadic functions is similar to that for monadic functions. Consider a dyadic function *F* with syntax *Z*←*A F B*. The derivative of *F* with respect to *B*, denoted *A F*ƀ*B*, must satisfy the conformability constraint suggested by the expression *B*+*A F B*. The result is the derivative (or partial derivatives) with respect to *B* or its elements, keeping *A* constant. The result is evaluated for the particular values of *A* and *B* presented.

A dyadic function can also be differentiated with respect to its left argument. In this case, the symbol ƀ is moved to the left (that is, it always appears between the function name and the independent variable). The rules are similar and in particular, for the outer derivative, its dimension is (⍴*A*∘.ƀ*F B*)=(⍴*A*),⍴*A F B*.

As an example, if functions for current density *J* and charge density *Q* are dyadic, with time on the left and space on the right, then the equation for conservation of charge, which appears in conventional notation as $\nabla\cdot J + \partial\sigma/\partial t = 0$, would appear in the new notation as 0=(T J+.ƀR)+TƀQ R. In a similar way, the substantive derivative, or "total derivative" of fluid mechanics $DN/Dt = \partial N/\partial t + V\cdot\nabla N$ can be written (TƀN R)+(T N∘.ƀ R)+.×V.

Some other interesting examples: The derivative of the maximum function A⌈ƀB is equal to B≥A. The derivative of the times function with respect to one argument is the other argument, that is (Aƀ×B)=B. If A is a vector then A×∘.ƀA is a diagonal matrix with the elements of A along the main diagonal.

Derivatives of Expressions

The notation so far is suitable for differentiation of a function with respect to an explicit argument. It may be desired to differentiate a function with respect to a global variable, or to differentiate an APL expression with respect to some variable in the expression. A slight generalization of the preceding notation is useful.

The derivative of an APL expression E with respect to a variable X is denoted either EƀX or XƀE. In the first case, if the expression is more complicated than a single niladic function, it must be parenthesized. The variable must be a variable name, perhaps indexed, but not a constant, function name, or expression. The expression might involve the independent variable explicitly, as in the identity (N×A*N-1)=(A*N)ƀA or (AƀA*N)=N×A*N-1, or perhaps as a global variable in a function, as in the identity ((A×B)ƀX)=((AƀX)×B)+A×BƀX; or perhaps in both roles.

Outer and inner derivatives of expressions with respect to global variables are denoted similarly, E∘.ƀX, X∘.ƀE, E+.ƀX, and X+.ƀE.

Note that since X must be a variable name rather than an expression, the following are syntax errors: EƀX+2; Eƀ(X); Eƀ-X; EƀX,0ρC←2; EƀX←3.

It might be thought that allowing both XƀE and EƀX could lead to ambiguities. However, the only case in which AƀB could be interpreted two ways is if both A and B are variable names, and in that case both interpretations, (A)ƀB and Aƀ(B), yield the same answer.

Implementation

The implementation of this notation appears to be straightforward, although perhaps demanding of space and time. The basic idea is that during execution of a function (or expression) being differentiated, a more complicated procedure is used. Every execution of a primitive function is accompanied by the calculation of the corresponding derivative. Thus, as the defined function is executed, line by line, not only are the values of the expressions calculated, but also their derivatives, always with respect to the original values of the argument in question. When the function finally returns, the value of the return variable is discarded and only the derivative is used.

Perhaps both the result of a given function, and its derivative, are required without re-executing the function, for example in case the function uses terminal input or makes changes in global variables. This is easily accomplished with the proposed notation. The last time the return variable is assigned, also assign some global variable. When the function returns, the global variable will retain the value of the function whereas the return variable will return the derivative.

Implementation in Detail

Consider a function F whose argument A is the independent variable, with respect to which the differentiation is being carried out. When the execution of F begins, the independent variable is considered to be accompanied by +∘.ƀA, that is, the outer derivative of A with respect to itself. Other variables, both global and local, are also considered to be accompanied by their partial derivatives with respect to *A*, in every case an array of zeros.

As the function *F* is executed, line by line, each execution of a primitive function is accompanied by the evaluation of the outer derivative of the result with respect to *A*. The chain rule of differentiation is used. Naturally, the derivatives so calculated will all be zero until *A* actually enters into the calculation.

If another defined function is called within the function being differentiated, this mode of operation continues and when that function terminates execution, not only the value of the return variable but also its outer derivative with respect to *A* are returned.

A given differentiation is completed when the function (or expression) being differentiated finally returns, or else is removed from the state-indicator stack by the right arrow. When the function returns, the return variable has both a value and its outer derivative with respect to *A*. The value is discarded and the derivative is returned. In the case of the derivative or inner derivative, appropriate operations are performed to extract the desired derivative. At the same time, all global variables have their accompanying derivatives erased.

When more than one derivative is being considered, a similar process applies. Variables are augmented with both the first and second derivative with respect to *A*.

Derivatives with respect to other variables may be encountered while any given derivative is pending. In that case, not only the derivatives with respect to each of the independent variables, but also cross partial derivatives have to be calculated. High-order derivatives of any order can be calculated recursively by using only the formulas for first-order derivatives.

While a derivative is being calculated, the normal APL environment should probably be changed in a few ways, that can be detected while the function is executing, or while it is suspended. The user should have a mechanism (perhaps a new ⌶ function) to determine how many derivatives of what order, are in progress, and (perhaps by changes to the)*SI* command) how far down the *SI* stack the derivatives were encountered. Also, the derivative of any expression *E* with respect to the Nth independent variable is denoted by *N*⊧*E*, or for the outer derivative, *N*∘.⊧*E*, or for the inner derivative, *N*+.⊧*E*. This feature allows functions to branch according to the values of derivatives, and also allows the display of derivatives for debugging.

Needless to say, much space and time will be saved by adopting the strategy of not actually storing derivatives that are identically zero. Thus, if execution is suspended while a derivative is pending, independent calculations can be carried out without necessarily calculating all the derivatives which end up equal to zero anyway.

Unresolved Problem

The derivative of every primitive function in APL can be written in terms of other primitive APL functions by use of the chain rule, with one exception. The exception is the scalar function !. The derivative of the gamma function cannot be expressed in terms of other implemented APL primitives. Furthermore, it is not known whether derivatives of arbitrary orders can be calculated through a recursion formula.

PERMUTATIONS

Michael M. Halpern
New College, Sarasota, Florida
and
IBM Scientific Center, Philadelphia, Pennsylvania

This article develops some concepts from elementary permutation theory in *APL*, then discusses the *APL* functions that permute their arguments. The design is thus twofold: *APL* as means, then as object, of exposition.

INTRODUCTION TO PERMUTATIONS

If two vectors contain the same elements, but not necessarily in the same order, they are called permutations of each other. In particular, a vector *V* which is a permutation of ⍳⍴*V* is said to be a permutation vector of order ⍴*V*. For example, '*STAR*' and '*TARS*' are permutations of each other related by the permutation vectors 2 3 4 1 and 4 1 2 3, i.e., '*STAR*'[2 3 4 1] ↔ '*TARS*' and '*TARS*'[4 1 2 3] ↔ '*STAR*'.

The matrix containing all permutation vectors of order *N* has !*N* rows and *N* columns. Indexing a vector of length *N* with such a matrix produces all of its permutations.

Generation of Permutation Vectors

A crude and obvious algorithm for generating all permutations of ⍳*N* is used in the sample function *PERMUTE*. It is non-iterative, non-recursive, straightforward, and quite inefficient: (*TIME* gives the elapsed time since its last call; measures of time and space are of course dependent upon the particular implementation used)

```
    ∇ P←PERMUTE N
[1]   P←1+(N⍴N)⊤¯1+N*N
[2]   P←⍉(∧/∨⌿P∘.=⍳N)/P
    ∇

      PERMUTE 3
 1 2 3
 1 3 2
 2 1 3
 2 3 1
 3 1 2
 3 2 1

    TIME
0.050
    A←PERMUTE 5
    TIME
8.833
    )ERASE A
    A←PERMUTE 6
WS FULL
PERMUTE[1] P←1+(NρN)⊤¯1+ιN*N
                        ∧
```

The function *PER* employs a wiser scheme exemplified by the following analysis of the case *N*←→4: given any permutation vector of order 3, say 2 1 3, we can form 4 permutation vectors of order 4 with (ι4)⌽4 4ρ2 1 3,4. Notice that *PER* works in both origins, and its output can be read either horizontally or vertically:

```
    ∇ P←PER N
[1]    P← 1 1 ρ1
[2]    →(N=1)/0
[3]    P←((!N),N)ρ((PER N-1),N)[;(ιN)⌽(N,N)ριN]
    ∇

      PER 3
 1 3 2
 3 2 1
 2 1 3
 2 3 1
 3 1 2
 1 2 3

      TIME
0.017
      K←PER 5
      TIME
0.067
      K←PER 6
      TIME
0.200
```

Distance and Parity

Given permutation vectors *A* and *B* of the same order, the distance between *A* and *B* is defined as the minimum number of transpositions (i.e. reversals of pairs) of adjacent elements required to transform *A* into *B*. For example, 2 4 1 3 is of distance 3 from 1 4 2 3:

```
2  4  1  3
2  1  4  3
1  2  4  3
1  4  2  3
```

The distance between *A* and ⍳⍴*A*, called the disorder of *A*, is given by

```
    +/,(A∘.<A)∧(⍳⍴A)∘.≥⍳⍴A
↔  +/1 1⍉+\A∘.<A
```

If *DIST* is a function which gives the distance between two permutation vectors, then *A DIST B* ↔ *A*[*C*] *DIST B*[*C*], for any permutation vector *C* of the same order as *A* and *B*. In particular,

```
    A DIST B
↔  A[⍋B]DIST B[⍋B]
↔  A[⍋B]DIST ⍳⍴B
```

allowing use of the disorder function already given.

The following definition gives the disorder of a vector permutation, or of the vectors across the last dimension of an array of permutations:

```
     ∇ Z←DISORDER A
[1]    Z←+/((⍳⍴⍴A),⍳⍴⍴A)⍉+\A∘.<A
     ∇
```

A permutation is said to be of even or odd parity, depending upon whether its disorder is even or odd. It is convenient to represent even parity by 1 and odd parity by ¯1, leading to the following definition:

```
     ∇ Z←PARITY A
[1]    Z←¯1*2|DISORDER A
     ∇
```

We can now define the determinant function *DET* on a square matrix *M*:

```
     ∇ D←DET M; P
[1]    P←PER (⍴M)[1]
[2]    D←(PARITY P)+.××/1 2 2⍉M[P;]
     ∇
```

```
      D
 1  4  2
¯1  1 ¯1
¯2  2  3

      DET 1 1↓D
5
     DET D
25
```

Properties of Permutations

If *W* is a permutation of *V*, then *V*⍳*W* is a mapping from *V* to *W*. For example, '*STAR*'⍳'*TARS*' ↔ 2 3 4 1 and '*STAR*'[2 3 4 1] ↔ '*TARS*'. In fact, *V* and *W* of equal length are permutations of each other if and only if *V*[*V*⍳*W*] ↔ *W* and *W*[*W*⍳*V*] ↔ *V*.

While this holds in all cases, least index does not produce a permutation vector unless there are no repeated elements in *V*, a condition indicated by (*V*⍳*V*)∧.=⍳⍴*V*. For example, '*POOL*' is a permutation of '*LOOP*' with '*LOOP*'[4 2 3 1] ↔ '*POOL*', but '*LOOP*'⍳'*POOL*' ↔ 4 2 2 1. The function which does set up a one-to-one correspondence between permutations is sometimes referred to as <u>progressive index</u>. It is similar to least index, except that each index of its left argument can appear in the result no more than once. The reader may anticipate the "Inverse Permutations" and "Double Upgrade" sections by analyzing this definition of *INDEX* for numeric vector arguments:

```
      ∇ I←V INDEX W
 [1]    I←((⍋V,W)⍳⍳⍴V)⍳(⍋W,V)⍳⍳⍴W
      ∇
```

This version of *INDEX* accepts vectors of either type:

```
      ∇ I←V INDEX W
[1]   I←((⍋V⍳V,W)⍳⍳⍴V)⍳(⍋V⍳W,V)⍳⍳⍴W
      ∇

      2 2 INDEX 2 2 2 2
1 2 3 3

      'LOOP' INDEX 'POOL'
4 2 3 1
```

If *V* is a permutation of ⍳⍴*V* then, according to the identities in the first paragraph, *V*[*V*⍳⍳⍴*V*] ↔ ⍳⍴*V* and (⍳⍴*V*)[(⍳⍴*V*)⍳*V*] ↔ *V*. Since *V* is a permutation vector, *V*⍳⍳⍴*V* is identical to ⍋*V*, and (⍳⍴*V*)⍳*V* is simply *V*. Thus *V*[⍋*V*] ↔ ⍳⍴*V* and (⍳⍴*V*)[*V*] ↔ *V*. For example:

```
      V←3 1 2 4

      ⍳⍴V
1 2 3 4

      V⍳⍳⍴V
2 3 1 4

      ⍋V
2 3 1 4

      V[⍋V]
1 2 3 4

      (⍳⍴V)⍳V
3 1 2 4

      (⍳⍴V)[V]
3 1 2 4
```

Random Permutations

A random permutation of order *N* is given by *N?N*:

```
      ⎕←P←6?6
6 3 5 4 1 2

      'EVOLVE'[P]
EOVLEV
```

RND produces a matrix of *R* random permutations of order *N*:

```
    ∇ M←R RND N
[1]   M←(R×N)?R×N
[2]   M←(R,N)⍴1+N|¯1+M[⍋⌈M÷N]
    ∇

      ⎕←P←3 RND 6
5 6 2 3 1 4
4 1 6 2 5 3
3 5 4 1 6 2

      'EVOLVE'[P]
VEVOEL
LEEVVO
OVLEEV
```

A convenient method of influencing a random process is the association of a weight with each entity involved. A vector of weights *W* can be represented in the following way:

```
    ∇ S←WEIGH W
[1]   S←1++/(⍳+/W)∘.>+\W
    ∇

      WEIGH 2 2 4 1
1 1 2 2 3 3 3 3 4
```

An element selected at random from *WEIGH W* has probability *W[I]÷+/W* of being *I*. In the following, weights control the frequency of each element of *⍳⍴W* in a random distribution:

```
      +/2 2 4 1
9

      ⎕←K←(WEIGH 2 2 4 1)[9?9]
3 2 4 1 2 3 3 3 1

      'FROG'[K]
ORGFROOOF
```

Here the weights represent the tendency of each element of *⍳⍴W* to appear toward the left of an otherwise random permutation vector:

```
      ((K⍳K)=⍳⍴K)/K
3 2 4 1
```

RANDOM restructures a matrix produced by *RND*:

```
    ∇ R←RANDOM S
[1]   R←(S[1],×/1↓S)⍴(S[1] RND S[2])∘.+S[3]⍴0
    ∇

      'APL *'[RANDOM 6 5 4]
LLLLPPPP****    AAAA
AAAALLLLPPPP****
****LLLL    PPPPAAAA
LLLLPPPP    ****AAAA
AAAAPPPPLLLL****
    PPPP****AAAALLLL
```

UPGRADE AND INVERSE PERMUTATIONS

Inverse Permutations

Since *V*[⍋*V*] ↔ ⍳⍴*V* for a permutation vector *V*, *V*[⍋*V*] is called an identity permutation, i.e., a vector permuted by it will remain unchanged. What about (⍋*V*)[*V*]? Using the identities of the previous paragraph and the associativity of indexing,

V[(⍋*V*)[*V*]] ↔ (*V*[⍋*V*])[*V*] ↔ (⍳⍴*V*)[*V*] ↔ *V*.

This can be true only if (⍋*V*)[*V*] ↔ ⍳⍴*V*. Thus *V* and ⍋*V* are inverse in the sense that indexing by one and then by the other is an identity function. In particular, if *V* is a permutation of *W*, then *V INDEX W* ↔ ⍋*W INDEX V*. Thus ⍋ generates the inverse of any permutation vector:

```
      V
3 1 2 4

      ⍋V
2 3 1 4

      (⍋V)[V]
1 2 3 4

      'EAST'[V]
SEAT

      'SEAT'[⍋V]
EAST

      'LOOP' INDEX 'POOL'
4 2 3 1

      ⍋'POOL' INDEX 'LOOP'
4 2 3 1
```

Double Upgrade

Upgrade and downgrade always produce permutation vectors, a property useful in explaining double upgrade: For any vector *X*, ⍋(⍋*X*) ↔ (⍋*X*)⍳⍳⍴*X*, meaning ⍋⍋*X* finds the index of each element of ⍳⍴*X* in ⍋*X* or, equivalently, the progressive index (see page 4) of *X*[⍳⍴*X*] ↔ *X* in *X*[⍋*X*]. Thus double upgrade ranks the elements of its argument. If *X* is a permutation vector, ⍋⍋ is clearly an identity function since ⍋ is then self-inverse:

```
      ⍋K←10 6 8 11
2 3 1 4

      ⍋⍋K
3 1 2 4

      ⍋⍋1 3 4 2
1 3 4 2

      ⍋⍋3 2 1 2
4 2 1 3
```

Double upgrade can be used to <u>mesh</u> (interweave) two vectors, or two arrays along a coordinate:

```
      VOWELS←'AIO'
      CONSONANTS←'RNBWS'
      (VOWELS,CONSONANTS)[⍋⍋2 1 1 2 2 1 2 2]
RAINBOWS

      ⍋⍋2 1 1 2 2 1 2 2
4 1 2 5 6 3 7 8
```

A <u>cyclic</u> mesh:

3-way:

```
      3|¯1+⍳12
0 1 2 0 1 2 0 1 2 0 1 2

      ⍋3|¯1+⍳12
1 4 7 10 2 5 8 11 3 6 9 12

      ⍋⍋3|¯1+⍳12
1 5 9 2 6 10 3 7 11 4 8 12

      V←'ADGJ','BEHK','CFIL'
      V[⍋⍋3|¯1+⍳12]
ABCDEFGHIJKL
```

```
N-way:            V[⍋|⍋N|¯1+⍳ρV]  ↔  ,⍉(N,(ρV)÷N)ρV
((ρV)÷N)-way:      V[⍋N|¯1+⍳ρV]  ↔  ,⍉(⌽N,(ρV)÷N)ρV
```

The Relationship Between Upgrade and Downgrade

Since upgrade and downgrade order repeated elements in the same direction (from left to right), ⍋V ↔ ⌽⍒V only if all the elements of V are unique. Thus (⍋V)∧.=⌽⍒V tests for this condition, and is equivalent to (V⍳V)∧.=⍳⍴V if V is numeric.

A general relationship is

```
⍋V ↔ ⍒K-V
```

for any scalar K, and in particular

```
⍋V ↔ ⍒-V .
```

This is conceptually related to the duality of dyadic ⌈ and ⌊ with respect to -. Furthermore, if we define functions F and G so that

```
         F V ↔ V[⍋V]
and      G V ↔ V[⍒V]
```

then F and G are dual with respect to - :

```
F V ↔ -G -V
V[⍋V] ↔ V[⍒-V] ↔  -(-V)[⍒-V].
```

Another case is

```
⍋V ↔ ⍒(⌈/V)-V ,
```

which becomes ⍋L ↔ ⍒~L for a logical vector L.

While ⍋⍋ is an identity function on a permutation vector P, ⍒⍒P ↔ ⌽⍋⍒P ↔ ⌽⍋⍋(1+⍴P)-P ↔ ⌽(1+⍴P)-P. Interestingly, quadruple downgrade is such an identity:

```
⍒⍒⍒⍒P ↔ ⌽(1+⍴P)-⌽(1+⍴P)-P ↔ (1+⍴P)-(1+⍴P)-⌽⌽P ↔ P.
```

Upgrade of Matrices and Character Vectors

The elements of a character vector can be arranged in lexical order by upgrading their numerical encoding:

```
      AB←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      Y←'GARDEN'

      AB⍳Y
7 1 18 4 5 14

      Y[⍋AB⍳Y]
ADEGNR
```

What does "upgrade of a matrix" mean? Two alternative interpretations are presented here. *ROWGRADES* upgrades the rows of a matrix independently of one another, while *SORTBYROW* simply upgrades a vector containing the decoding (⊥) of each row:

```
      ∇ R←ROWGRADES M; N; U
[1]     N←¯1↑⍴M
[2]     U←⍋,M
[3]     R←(⍴M)⍴1+N|¯1+U[⍋⌈U÷N]
      ∇

        A
8 6 1 3
1 3 2 5
1 2 3 4

        ROWGRADES A
3 4 2 1
1 3 2 4
1 2 3 4

        1 1 2⍉A[;ROWGRADES A]
1 3 6 8
1 2 3 5
1 2 3 4

      ∇ R←SORTBYROW M
[1]     R←⍋(⌈⌿M)⊥⍉M
      ∇

        SORTBYROW A
3 2 1

        B
FIRE
LAKES
LAKE
FUR
RAIN

      ∇ A←ALPH M; CHARS
[1]    CHARS←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[2]    A←SORTBYROW CHARS⍳M
      ∇

         B[ALPH B;]
FIRE
FUR
LAKE
LAKES
RAIN
```

THE PRIMITIVE PERMUTATION FUNCTIONS

The monadic and dyadic forms of ⌽ and ⍉ all permute their arguments, except when the left argument of ⍉ is not a permutation vector. We have already dealt with the most general permutation function, indexing, and with ⍋ and ⍒, which not only conserve, but create, permutation vectors.

The Monadic Permutation Functions

The symbols ⍉, ⌽, and ⊖ are mnemonic in that the straight lines represent axis orientations in a matrix argument. Specifically, transposition is rotation through a third dimension around the major diagonal. Thus ⍉ allows determination of symmetry with respect to this diagonal. For example, a function α is commutative if its multiplication table *M* (of the form *X*∘.α*X* for domain *X*) is identical to ⍉*M*. ⌽ and ⊖ analogously check for bilateral symmetry.

Eight permutations of a matrix are obtainable using transposition and reversal:

```
    M           ⍉⊖M        ⊖⌽M        ⊖⍉M
                ⌽⍉M        ⌽⊖M        ⍉⌽M

  1 2 3        7 4 1      9 8 7      3 6 9
  4 5 6        8 5 2      6 5 4      2 5 8
  7 8 9        9 6 3      3 2 1      1 4 7


   ⍉M           ⌽M        ⊖⌽⍉M        ⊖M
                          ⍉⌽⊖M

  1 4 7        3 2 1      9 6 3      7 8 9
  2 5 8        6 5 4      8 5 2      4 5 6
  3 6 9        9 8 7      7 4 1      1 2 3
```

Rotate

For a vector *V* and integer *R*,

```
R⌽V ↔ V[1+(⍴V)|R+¯1+⍳⍴V]
    ↔ V[⍋(⍳⍴V)≤(⍴V)|R]
```

The second form suggests this method of suffix rotation:

```
((K-1)↑V),1⌽(K-1)↓V ↔ V[⍋K=⍳⍴V]
```

```
      4=ι6
0 0 0 1 0 0

      ⍋4=ι6
1 2 3 5 6 4

      'SAMPLE'[⍋4=ι6]
SAMLEP

      'SAM',1⌽'PLE'
SAMLEP
```

Right rotation is the inverse permutation of left rotation:

```
((K-1)↑V),¯1⌽(K-1)↓V ↔ V[⍋⍋K=ιρV]
```

In either of the above substitute ≠ for =, or ⍒ for ⍋, to rotate the prefix in the opposite direction.

Transpose in Terms of Reshape

The connection between reshape and transpose is stronger than one might suspect. Since *V*ρ*A* ↔ *V*ρ,*A*, reshape consists of peeling off successive elements of a vector and arranging them according to a coordinate precedence convention, viz., the last varying most rapidly, the first most slowly, etc. The same convention is used in printing arrays. A monadic transpose simply reverses this order:

```
      2 2 2ρι8                  ⍉2 2 2ρι8
1 2                       1 5
3 4                       3 7

5 6                       2 6
7 8                       4 8
```

The left argument of a dyadic transpose specifies the coordinate precedence scheme to be used in reshaping the right argument. In *V*⍉*A*, the *I*th coordinate is made to vary *V*[*I*]th most slowly:

```
      3 1 2⍉2 2 2ρι8
1 5
2 6

3 7
4 8
```

A 1 1 transpose is also easy to understand: the first and second coordinates vary at the same rate, so the major diagonal is extracted.

Transposition as a Permutation

If *P* is a permutation vector, then *P*⍉*A* permutes *A*. *P* specifies the destinations in *P*⍉*A* of successive coordinates of *A*. As an index of a vector, *P* would give the sources in *V* of *V*[*P*]'s successive elements. These two methods of specifying a permutation are in fact inverse, and are thus related by ⍋. This is exploited in the following representation of transposition, in which *P* is a permutation vector:

```
     ∇ T←P TRANSPOSE A; K; N
[1]    K←⍴A
[2]    N←K[⍋P]
[3]    T←N⍴(,A)[1+K⊥(N⊤¯1+⍳×/K)[P;]]
     ∇
```

Applications of Transpose

The function *JOIN* connects two arrays of equal rank along the last coordinate. A function which connects matrices along the row coordinate is simply the dual with respect to ⍉ of *JOIN*:

```
     ∇ J←M JOIN N; R
[1]    R←¯1↓(⍴M)⌈⍴N
[2]    J←((R,¯1↑1,⍴M)↑M),(R,¯1↑1,⍴N)↑N
     ∇
```

```
      A                         B
AAAA                      BB
AAAA                      BB
AAAA
AAAA

      A JOIN B                   ⍉(⍉A) JOIN ⍉B
AAAABB                    AAAA
AAAABB                    AAAA
AAAA                      AAAA
AAAA                      AAAA
                          BB
                          BB
```

Arrays *M* and *N* of the same rank may be joined along any coordinate *I* by

```
                K←⍋I=⍳⍴⍴M
                K⍉((⍋K)⍉M)JOIN(⍋K)⍉N .
```

By such a use of transposition, an algorithm based upon a specific coordinate system may often be generalized to any permutation of that system.

PERMUTATION REPRESENTATION AND GENERALIZATION

Permutation Matrices

A permutation vector `P` corresponds to a matrix of the form `(ιρP)∘.=P`, or `((ιρP)∘.=ιρP)[;P]`. Thus the permutation matrix representing 4 2 1 3 is

```
0 0 1 0
0 1 0 0
0 0 0 1
1 0 0 0
```

Permutation matrices corresponding to inverse permutation vectors are themselves matrix inverses or, equivalently, transposes. This is because each row and each column of a permutation matrix has exactly one 1. Thus, if `M ↔ (ιρP)∘.=P`, then `M+.×⍉M ↔ M∨.∧⍉M`. Since rows are identical if they have a 1 in the same column, `M∨.∧⍉M ↔ M∧.=⍉M`. Since the rows of `M` are distinct, `M∧.=⍉M ↔ (ι1↑ρM)∘.=ι1↑ρM ↔ (ιρP)∘.=ιρP`, the identity matrix, which is the permutation matrix of the identity permutation. In particular, a permutation vector `P` is self-inverse, i.e., `P[P] ↔ ιρP` or `P ↔ ⍋P`, if its permutation matrix is self-transpose.

Generalizations of Permutation

The preceding sections have been concerned with permuting the elements of a vector. Two generalizations, permuting <u>segments</u> of a vector and permuting a general array <u>along a coordinate</u>, are offered below.

The right argument of `MIX` specifies how a global vector `V` is to be segmented by giving the lengths of successive "blocks." The left argument supplies the desired permutation of these blocks. The arguments must be of equal length, and the sum of the right argument must be equal to the length of the global vector `V` whose segments are permuted (see page 6 for `WEIGH`):

```
    ∇ M←P MIX L
[1]   M←V[⍋P[WEIGH L]]
    ∇


          V
SUM ERGO COGITO

      3 2 1 MIX 3 6 6
COGITO ERGO SUM
```

Note that *V* could be made an argument of *MIX* by catenating *P* and *L* to form a single argument.

In these examples permutation is achieved by indexing, but the notation for indexing in *APL*\360 makes it awkward to permute (along some coordinate) an array whose rank is variable. The function *PERM* applies a permutation vector *P* to the last coordinate of an array *A*:

```
     ∇ B←P PERM A; K; L
[1]    K←×/¯1↓ρA
[2]    L←¯1↑1,ρA
[3]    B←(ρA)ρ((K,L)ρA)[;P]
     ∇

      ⎕←A←2 2 3ρ'RPETMUIATSON'
RPE
TMU

IAT
SON
      2 3 1 PERM A
PER
MUT

ATI
ONS
```

An alternate method, which requires a <u>numeric</u> array, is based upon permutation matrices. *NPERM* applies a permutation vector *P* to the last coordinate of a numeric array *N*:

```
     ∇ M←P NPERM N
[1]    M←N+.×(ιρP)∘.=P
     ∇

      3 1 2 NPERM 2 3ρι6
3 1 2
6 4 5
```

REFERENCES

Falkoff, A.D. and K.E. Iverson, <u>APL/360 User's Manual</u>, IBM Corp., GH20-0906, 1968; a complete description of the APL/360 system.

Iverson, K.E., "Formalism in Programming Languages," <u>Communications of the ACM</u>, February 1964; a source of APL identities, easily translatable into the present APL language.

XEROX FINANCIAL PLANNING AND CONTROL SYSTEM IN APL

By J. W. ROUGH

The ISG Planning and Control System is a system of models and data bases which has been under development at Xerox since mid 1972. It is being developed by Management Sciences for the Information Services Group (ISG), the division responsible for marketing, finance and distribution in Xerox within the continental United States. It is a functionally oriented system which is being implemented in a piecemeal fashion, each section oriented toward the needs of the individual users, but all components consistent with one another. The system will have the capability to forecast, evaluate and communicate important business factors, including problems, options and strategies. The use of APL is central to this general objective, because it allows a non-programmer to take full advantage of the capabilities of the system, with turnaround times of only a few minutes.

Exhibit I is an overall diagram of the ISG Planning and Control System showing the components of the system and their simplified relationships. Each module is a user oriented subsystem which will be pressed into operation as soon as it is developed. CAF (Computer Assisted Forecasting) is the primary model in the system, simulating product movements for each month considering Marketing's estimate of the demand for machines, manufacturing's estimate of the supply of machines, and the constraints of Xerox' distribution network.

An unconstrained model is the Demand Forecast module which predicts product demand from an analysis of the current on-lease population and the potential market. It establishes the most reasonable plan of reaching Xerox' revenue goals in terms of cancellations, new business and trades. This model feeds its three basic results to CAF where a detailed plan, including for example, Branch Inventories and Machines to be Refurbished, is determined. CAF may also be used to allocate the detailed plan downward to the region and branch levels.

The Data Base module is the information source for the system. It provides data directly to the models or through the Factors Forecast module, and to the users through reports or special purpose analytical routines.

Each subsystem is discussed in greater detail following a description of the system objectives.

OBJECTIVES

The objectives of the ISG Planning and Control System are to provide the decision maker with:

- Accurate and consistent historical information.
- A method of forecasting and storing the forecasts which is consistent among users in various functional areas.
- A tool to quickly pinpoint problem areas and help determine the financial impacts of options and strategies.
- A tool to assess the need for additional information and its dollar value.
- A greater understanding of the assumptions and approximations that are made during the forecasting process, and how these affect the accuracy of the forecast.
- The ability to clearly communicate aspects of the business through reports individual to the situation.
- The flexibility to incorporate additional models and management experience within the system.
- A system which is clearly definable in terms of data responsibility for inputting, validating and updating, and which has adequate data security against unauthorized access.

DATA BASES

This module of the system is a series of data files which contain plan and historical data for activity, revenue, expenses, manpower and other information areas. They contain 2 years historical data for each of eighty branches, by product line, by individual account, plus assorted summary levels. These data bases are exceptionally large for APL, about 10-20 megabytes each, and represent Xerox' first experience interfacing in APL with the COBOL systems environment. The five data bases and their interaction with the rest of the system are shown in more detail in Exhibit #2.

Each data base has a number of modularized routines associated with it which retrieve and restructure the data into matrices that may be used in other applications or to perform various analyses. Also associated with the data bases are assorted reporting and analytical packages which develop the data for reporting purposes or for feeding into other models. An example might be a statistical package to project outright cancellation rates and feed the projections to CAF, the constrained forecasting model.

The many programs connected with each data base are contained with brief descriptions in an APL library for Xerox users so that someone unfamiliar with the data base may have the information available to him. For example, a user desiring to look at net additions for a certain branch as compared with all branches in a certain month, would look first in the Xerox Activity Data Base Library. There he would quickly determine if a suitable report was available, or which access functions could be used to retrieve the data he desired in matrix format. There are functions described in the library of each data base which provide access across almost any possible grouping of information. However, for specific purposes the files may be accessed directly by the user since the structure of all the data bases will be similar and conceptually easy to visualize (Exhibit 3).

CAF (COMPUTER ASSISTED FORECASTING) MODEL

This module of the system simulates product movements, expenses, revenue, and manpower for the branch, region or Group. Exhibit 2 pictures how the components of CAF relate to the individual data bases and the routines which generate the factors. Historical data from the data bases establishes the inventories and backlogs within CAF, while the analyst inputs the machine supply and demand schedules. The inputs may be entered on a trial and error basis, a procedure that would probably be used in branch planning for example, or they may be obtained from the Demand and Supply Forecast modules.

The Demand Forecast module supplies CAF with a schedule of net new business orders and information regarding trade patterns and outright cancellations. CAF then develops this basic Demand information in conjunction with the data bases and the supply input to simulate month to month product movements for the company as a whole. The model considers pipelines, inventories and system restraints together with assumptions made interactively by the user, to forecast a wide variety of activity variables. Exhibit #4 is a diagram examining the simulation process of CAF in greater detail. The final results of the activity simulation process may be used with new sets of assumptions to determine separate revenue, manpower, and expense forecasts. For example, once an agreeable plan of product movements is established, it may be used as the activity basis for generating a number of revenue forecasts for different price plans.

Besides developing the National plan, CAF may also be used with independently determined factors to allocate the plan to the regions, and the branches. Because the same model is used to develop telescoping forecasts, differences of opinion and discrepancies in logic can readily be isolated and discussed. The factors and rates may be compared directly in determining what is, and what is not, realistic. Areas where more information is needed become apparent as users isolate certain factors to be the areas of disagreement and uncertainty. Also, because CAF is conversant with the Demand Forecast module, the logistics of sales, distribution, and service, which are traced in CAF, can be meshed with the marketplace orientation of the goals developed by top management. In this way management goals and strategies may be logically and consistently structured into a specific plan for the company.

DEMAND FORECAST

This module will generate unconstrained demand forecasts through consideration of the market, the competition, different marketing strategies and management goals. The concept of the model is to provide the planner with flexibility, as well as a set of reasoned possibilities and mathematically sound relationships.

The user will be able to use statistical packages to analyze past data dealing with copy average distributions, customer trade patterns, seasonal and trend flunctuations and other phenomena relating to new business, trades and removals. Given management goals in terms of gross revenue, analysts will be able to use their experience as well as various mathematical routines to study different strategies, including price plans and the introduction of new products. The demand forecasts developed by the model will be a schedule of net additions for each machine, broken up into net new business orders, trades, and cancellations. These forecasts will be used in the CAF portion of the ISG system to develop more detailed forecasts which are constrained by the production capabilities and the distribution network.

SUPPLY FORECAST

This module is the input to CAF which predicts the supply of machines available to the system. Generally, a model will not be necessary to predict this input, at least within ISG, because the supply schedule from manufacturing is known. Also, many users of CAF will want to run the model using phony supply numbers in order to determine, for example, how many machines will be required from manufacturing, or the supply of machines already within the pipelines.

FACTORS FORECAST

The factors module in the system is designed to help the analyst use the forecasting models. It provides him with statistical tools that predict the factors needed by the models and evaluate the forecasts that the models develop. As an example, CAF requires a factor, "percent of removals which are returned to manufacturing". This factor is used in determining how many refurbishable machines are available to the system. In addition to being able to enter this percent directly, the Factors module also gives the analyst an opportunity to use the past history of this factor in predicting its new values. Through the use of exponential smoothing, or last quarter's average, or by extrapolating along some regression curve, the planner can develop accurate and flexible estimates with little effort. Also, by tracking the various factors and their predictions, other simpler forecasting models can be developed and the accuracy of the models can be improved.

PLAN DATA BASE (PDB)

This module of the system connects the ISG Planning and Control Model to the other operating systems in the company. It is essentially a data storage area where the year's entire operating plan resides while being validated and checked. For example, if a new product introduction slips a month, or if a new branch is created, the plan in PDB will be adjusted. Once a month these adjustments are fed into Xerox operating systems.

CURRENT STATUS

At this writing the ISG Planning and Control System consists of a completed Data Base module, a completed Plan Data Base module, and a series of independent financial models written previously in both APL and Fortran. The data bases contain comprehensive expense, revenue, manpower and activity information which provide a solid historical foundation for the forecasting requirements of the various functional areas. The value of having this data available in APL has greatly expanded the extent of this portion of the system, and refinements and modifications continue to further extend the project scope.

Work on models within the system is progressing in a manner which satisfies the immediate needs of the funding departments. These functional areas already have substantial models which partially fulfill both their requirements and the modeling goals outlined above. Management Sciences is coordinating these two ends in order to meet the short term needs of the financial areas as well as the systems objectives expressed earlier.

ISG PLANNING & CONTROL SYSTEM

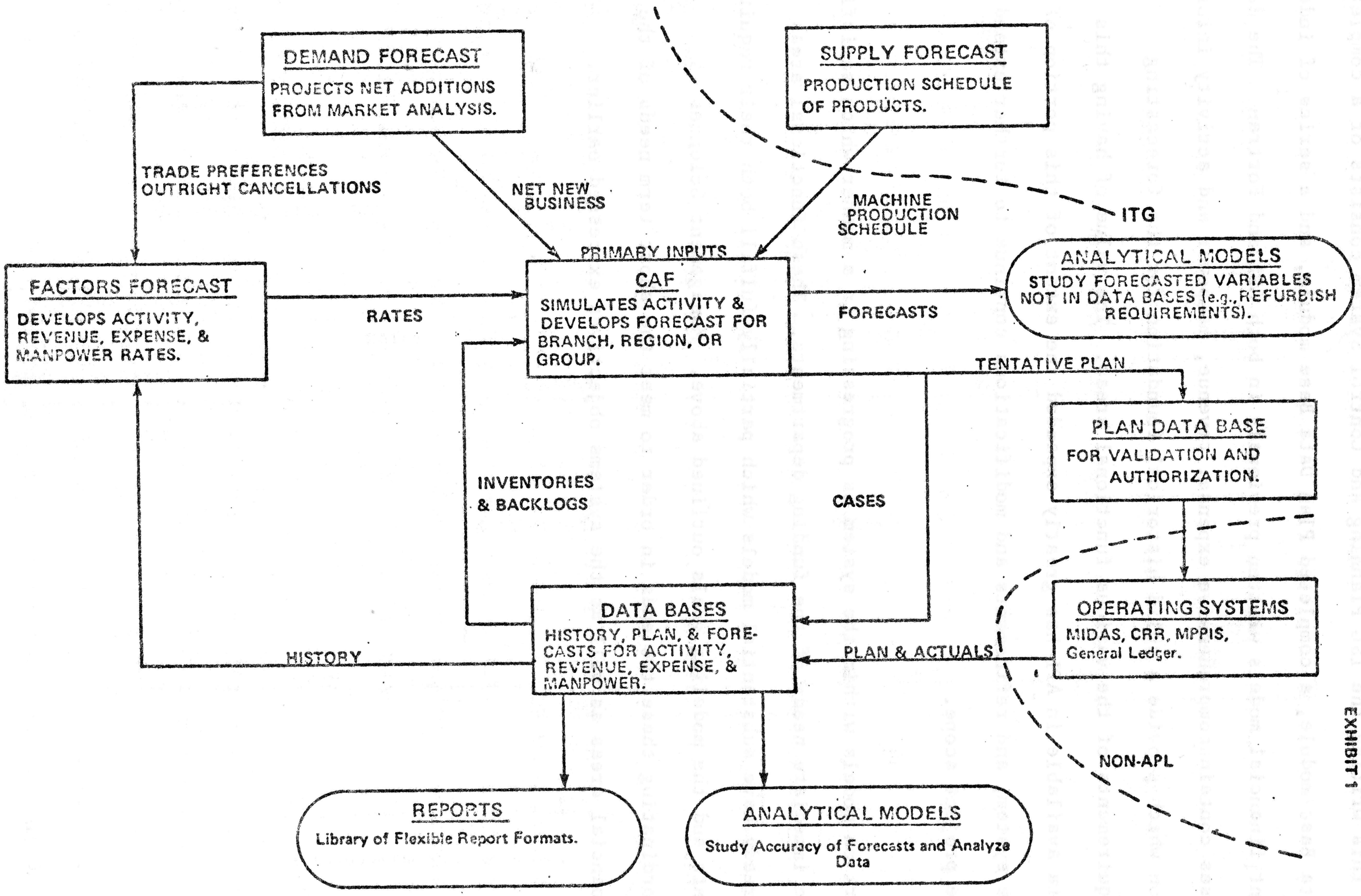
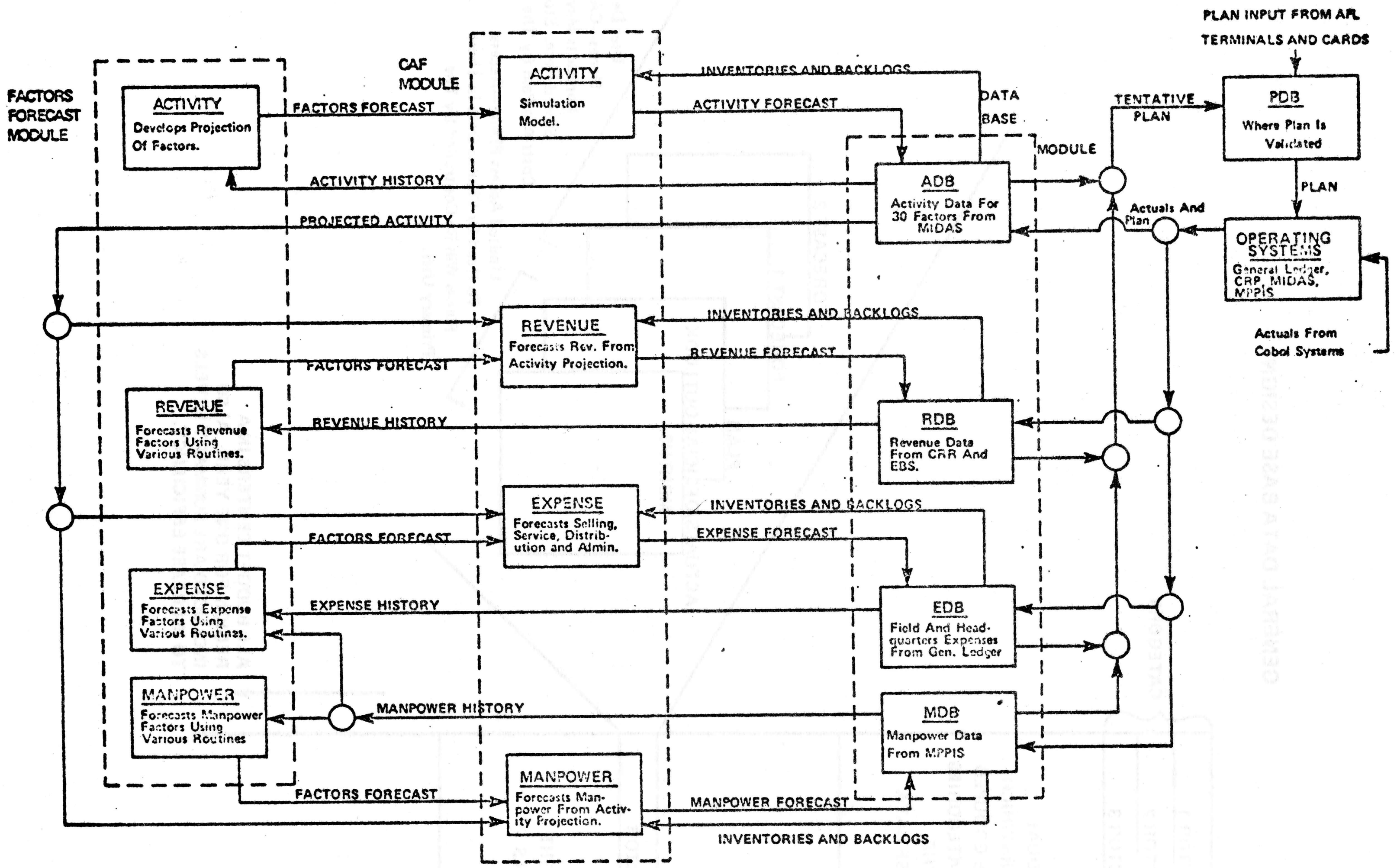
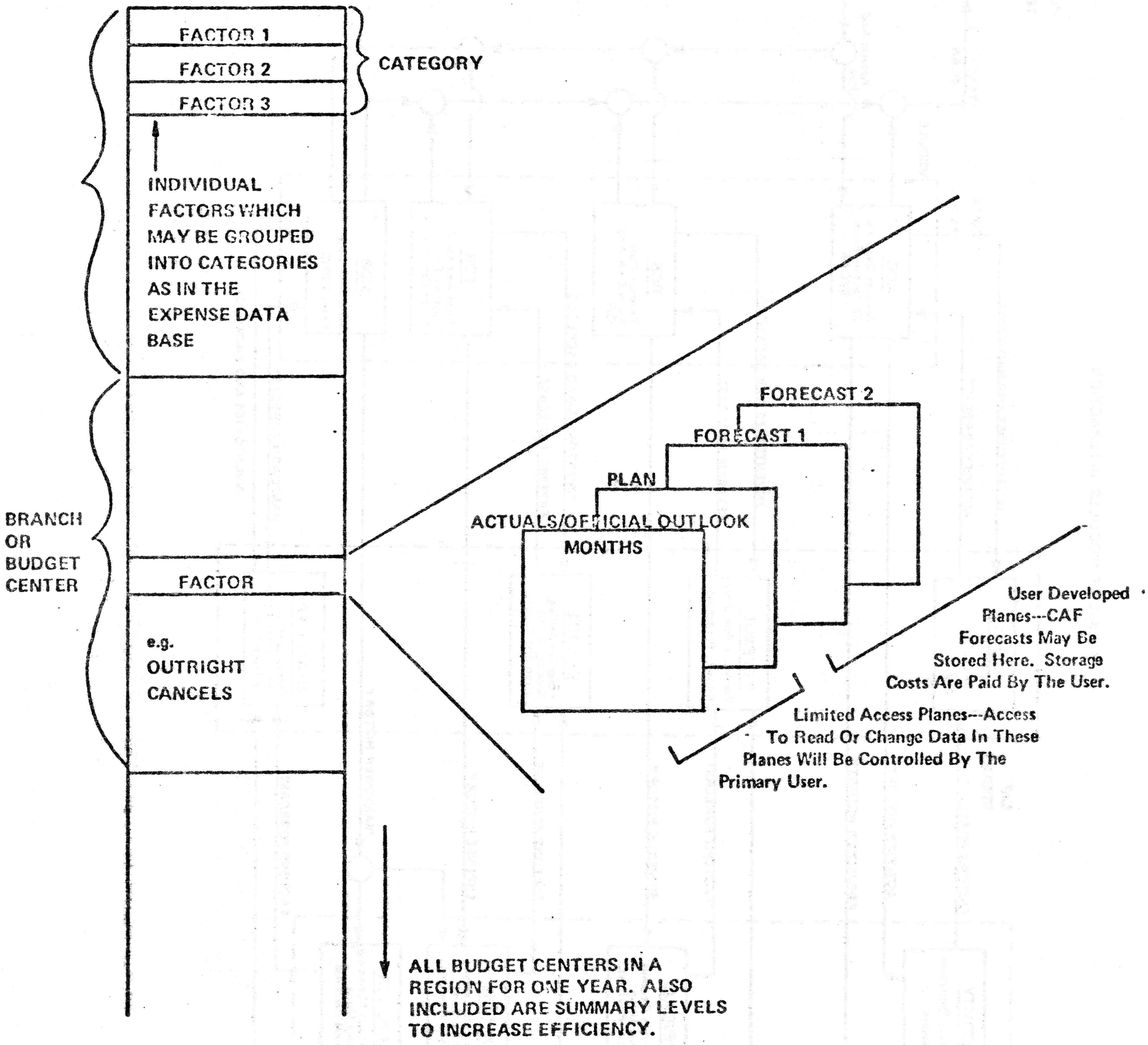

DEMAND FORECAST
PROJECTS NET ADDITIONS FROM MARKET ANALYSIS.
SUPPLY FORECAST
PRODUCTION SCHEDULE OF PRODUCTS.
TRADE PREFERENCES
OUTRIGHT CANCELLATIONS
NET NEW BUSINESS
MACHINE PRODUCTION SCHEDULE
ITG
PRIMARY INPUTS
FACTORS FORECAST
DEVELOPS ACTIVITY, REVENUE, EXPENSE, & MANPOWER RATES.
RATES
CAF
SIMULATES ACTIVITY & DEVELOPS FORECAST FOR BRANCH, REGION, OR GROUP.
FORECASTS
ANALYTICAL MODELS
STUDY FORECASTED VARIABLES NOT IN DATA BASES (e.g.,REFURBISH REQUIREMENTS).
TENTATIVE PLAN
PLAN DATA BASE
FOR VALIDATION AND AUTHORIZATION.
INVENTORIES & BACKLOGS
CASES
OPERATING SYSTEMS
MIDAS, CRR, MPPIS, General Ledger.
DATA BASES
HISTORY, PLAN, & FORECASTS FOR ACTIVITY, REVENUE, EXPENSE, & MANPOWER.
HISTORY
PLAN & ACTUALS
NON-APL
REPORTS
Library of Flexible Report Formats.
ANALYTICAL MODELS
Study Accuracy of Forecasts and Analyze Data

EXHIBIT 1

DETAIL OF MODULES' INTERACTION


PLAN INPUT FROM APL TERMINALS AND CARDS
FACTORS FORECAST MODULE
ACTIVITY
Develops Projection Of Factors.
REVENUE
Forecasts Revenue Factors Using Various Routines.
EXPENSE
Forecasts Expense Factors Using Various Routines.
MANPOWER
Forecasts Manpower Factors Using Various Routines
CAF MODULE
ACTIVITY
Simulation Model.
REVENUE
Forecasts Rev. From Activity Projection.
EXPENSE
Forecasts Selling, Service, Distribution and Admin.
MANPOWER
Forecasts Manpower From Activity Projection.
DATA BASE MODULE
ADB
Activity Data For 30 Factors From MIDAS
RDB
Revenue Data From CRR And EBS.
EDB
Field And Headquarters Expenses From Gen. Ledger
MDB
Manpower Data From MPPIS
PDB
Where Plan Is Validated
OPERATING SYSTEMS
General Ledger, CRP, MIDAS, MPPIS
Actuals From Cobol Systems
TENTATIVE PLAN
PLAN
Actuals And Plan
FACTORS FORECAST
ACTIVITY HISTORY
PROJECTED ACTIVITY
INVENTORIES AND BACKLOGS
ACTIVITY FORECAST
FACTORS FORECAST
REVENUE HISTORY
INVENTORIES AND BACKLOGS
REVENUE FORECAST
FACTORS FORECAST
EXPENSE HISTORY
INVENTORIES AND BACKLOGS
EXPENSE FORECAST
MANPOWER HISTORY
FACTORS FORECAST
MANPOWER FORECAST
INVENTORIES AND BACKLOGS

GENERAL DATA BASE DESIGN

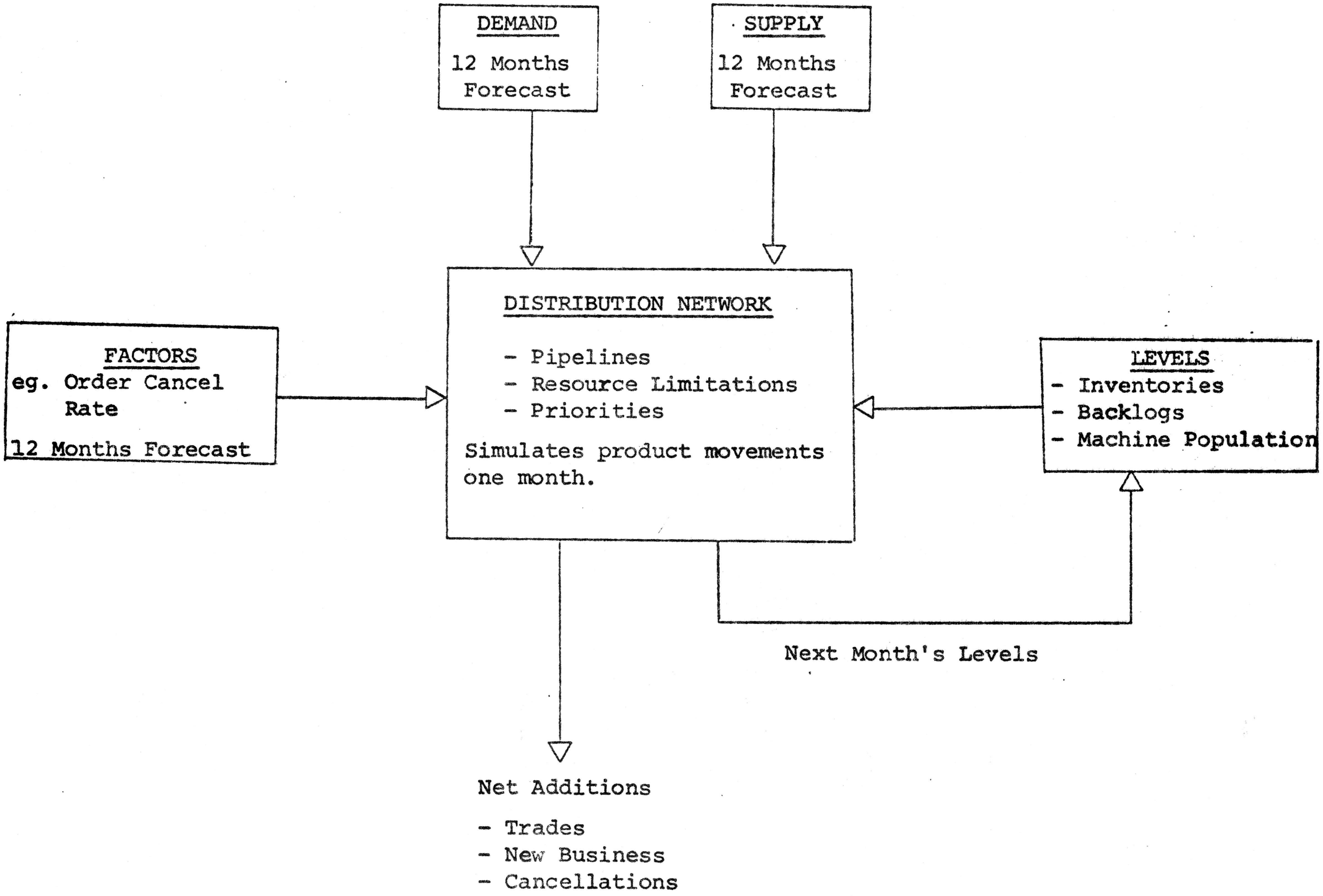

FACTOR 1
FACTOR 2
FACTOR 3
CATEGORY
INDIVIDUAL FACTORS WHICH MAY BE GROUPED INTO CATEGORIES AS IN THE EXPENSE DATA BASE
BRANCH OR BUDGET CENTER
FACTOR
e.g. OUTRIGHT CANCELS
FORECAST 2
FORECAST 1
PLAN
ACTUALS/OFFICIAL OUTLOOK
MONTHS

User Developed Planes---CAF Forecasts May Be Stored Here. Storage Costs Are Paid By The User.

Limited Access Planes---Access To Read Or Change Data In These Planes Will Be Controlled By The Primary User.

ALL BUDGET CENTERS IN A REGION FOR ONE YEAR. ALSO INCLUDED ARE SUMMARY LEVELS TO INCREASE EFFICIENCY.

CAF SIMULATION MODEL


DEMAND
12 Months Forecast
SUPPLY
12 Months Forecast
FACTORS
eg. Order Cancel Rate
12 Months Forecast
DISTRIBUTION NETWORK
- Pipelines
- Resource Limitations
- Priorities
Simulates product movements one month.
LEVELS
- Inventories
- Backlogs
- Machine Populatio
Next Month's Levels
Net Additions
- Trades
- New Business
- Cancellations

THE BROWNBOX DATA MANAGEMENT SYSTEM

By D. FLOWER

ABSTRACT

THIS SYSTEM CONSISTS OF A SET OF FUNCTIONS THAT GREATLY SIMPLIFY THE WRITING OF INFORMATION SYSTEMS. DATA CAN BE STORED IN A RECORD FORMAT WHERE EACH RECORD CONSISTS OF A GROUP OF FIELDS. FIELDS CAN BE EITHER CHARACTER OR NUMERIC AND EITHER FIXED OR VARIABLE LENGTH. THE PRESENCE OF ANY PARTICULAR FIELD IN A RECORD IS ENTIRELY OPTIONAL. ANY FIELD CAN BE INSERTED INTO OR DELETED FROM ANY RECORD AT ANY TIME. WHEN A FIELD IS NOT PRESENT IT TAKES UP NO SPACE IN THE RECORD.

THE SYSTEM AUTOMATICALLY MAINTAINS INDEXES OVER ALL FIELDS THE USER DECLARES AS INDEXED. AN INDEX ALLOWS THE USER TO RETRIEVE RECORDS BY SPECIFYING THE CONTENTS OF A FIELD IN THE RECORD. THE USER DOES NOT HAVE TO KNOW THE LOCATION OF THE RECORD. FOR EXAMPLE, A STUDENT'S RECORD COULD BE RETRIEVED MERELY BY SPECIFYING THE STUDENT'S NAME.

WHEN TWO OR MORE PEOPLE ARE SIMULTANEOUSLY WORKING WITH THE SAME SET OF FILES, THE SYSTEM MAINTAINS FILE INTEGRITY THROUGH THE USE OF A SOPHISTICATED INTERLOCK MECHANISM. A VARIANT OF THIS SAME MECHANISM PROVIDES THE USER WITH A BACKUP CAPABILITY - IF A MISTAKE IS MADE WHILE ALTERING A FILE, THE FILE CAN BE COMPLETELY RESTORED TO ITS ORIGINAL STATE SIMPLY BY TYPING 'OUT'.

SYSTEM SECURITY ALLOWS DETAILED CONTROL OF A USER'S ACCESS TO THE FILES. A USER CAN BE GIVEN READ AND/OR WRITE ACCESS ON A FIELD-BY-FIELD BASIS. THAT IS, A USER COULD BE GIVEN READ ACCESS TO SOME FIELDS IN A RECORD, WRITE ACCESS TO OTHERS, AND NO ACCESS AT ALL TO THE REMAINDER. DIFFERENT USERS CAN BE GIVEN DIFFERENT ACCESS PERMISSIONS.

EXAMPLES OF BROWNBOX USE

SUPPOSE WE HAVE A FILE OF INFORMATION ON STUDENTS. EACH STUDENT HAS A RECORD CONTAINING MANY FIELDS OF INFORMATION - NAME, ADDRESS, BIRTHDATE, CLASSES BEING TAKEN, MARKS, ETC. (SEE APPENDIX 1 FOR A FULL DISCUSSION OF THE STRUCTURE OF THIS FILE). WE WILL WANT TO RETRIEVE THE RECORDS FOR VARIOUS STUDENTS AND DISPLAY OR ALTER THE CONTENTS OF SELECTED FIELDS.

SUPPOSE WE WANT TO FIND THE HOME ADDRESS OF THE STUDENT WHOSE NAME IS 'JOHN STEVEN GREEN'. WE TYPE:

```
      GT ADR AT NAMID IS 'JS GREEN'
```

AND THE COMPUTER RESPONDS:

```
  34 ELM ST.; TORONTO 76; ONTARIO  .
```

THIS OPERATION IS ACTUALLY A COMBINATION OF TWO MORE ELEMENTARY OPERATIONS: FIRST WE RETRIEVE THE STUDENT'S RECORD AND THEN WE GET THE CONTENTS OF THE ADDRESS FIELD:

```
      NAMID IS 'JS GREEN'
  1  .
```

THE 'IS' FUNCTION RETRIEVES THE RECORD WHOSE 'NAMID' FIELD CONTAINS 'JS GREEN'. 'NAMID' IS A FIELD THAT CONTAINS THE INITIALS OF A PERSON'S FIRST TWO NAMES FOLLOWED BY A BLANK AND THE LAST NAME. SINCE THE SYSTEM INTERNALLY IDENTIFIES FIELDS ONLY BY THEIR FIELD NUMBERS, 'NAMID' MUST IN FACT BE A NUMERIC VARIABLE CONTAINING THE FIELD NUMBER - IN THIS CASE, 2. WE ARE SIMPLY USING 'NAMID' AS A MNEMONIC; WE COULD HAVE GOTTEN THE SAME RESULT AS ABOVE BY TYPING:

```
      2 IS 'JS GREEN'
  1  .
```

THE '1' RETURNED BY THE 'IS' FUNCTION IS AN INDICATION THAT THE SOUGHT-FOR RECORD HAS BEEN FOUND. IF THE RECORD IS NOT FOUND (I.E. THE FILE CONTAINS NO RECORD WHOSE 'NAMID' FIELD CONTAINS 'JS GREEN', THE FUNCTION RETURNS '0'.

HAVING RETRIEVED THE RECORD WITH THE 'IS' FUNCTION, WE NOW WANT TO GET THE CONTENTS OF THE ADDRESS FIELD, WHOSE MNEMONIC IS 'ADR':

```
      GT ADR
  34 ELM ST.; TORONTO 76; ONTARIO  .
```

THE 'GT' FUNCTION RETURNS THE CONTENTS OF THE FIELD SPECIFIED ON THE RIGHT.

NOW THAT WE HAVE EXPLAINED HOW TO RETRIEVE A RECORD WITH 'IS' AND HOW TO GET THE CONTENTS OF A FIELD WITH 'GT', WE CAN PUT BOTH OF THEM TOGETHER ON ONE LINE USING THE 'AT' FUNCTION:

```
          GT ADR AT NAMID IS 'JS GREEN'
      34 ELM ST.; TORONTO 76; ONTARIO   .
```

THE 'AT' FUNCTION JUST GLUES THE ITEM ON THE LEFT TO THE ITEM ON THE RIGHT AND DOES NOTHING ELSE.

HAVING GOTTEN THE ADDRESS, LETS NOW DISPLAY ALL THE CLASSES THIS STUDENT IS TAKING:

```
          AL CLASS
      MATH1
      CHEM1
      BIOLOGY8
      HIST5   .
```

THE 'AL' FUNCTION RETURNS A MATRIX CONTAINING THE VALUES OF ALL OCCURRENCES OF THE FIELD SPECIFIED ON THE RIGHT. IN THIS CASE THE 'CLASS' FIELD HAS FOUR OCCURRENCES CONSISTING OF THE ABOVE FOUR VALUES. (NOTICE THAT WE DID NOT HAVE TO RESPECIFY THE RECORD WITH ANOTHER USE OF 'NAMID IS 'JS GREEN'' - ONCE WE HAVE RETRIEVED A RECORD IT REMAINS IN MEMORY UNTIL WE RETRIEVE A DIFFERENT RECORD.)

IF, INSTEAD OF ALL CLASSES, WE JUST WISHED TO SEE THE THIRD CLASS, WE COULD HAVE TYPED:

```
          GT CLASS,3
      BIOLOGY8   .
```

IN GENERAL , WE HAVE TO SPECIFY WHICH OCCURRENCE OF A FIELD WE WANT. WHEN NO OCCURRENCE IS SPECIFIED, THE SYSTEM, BY DEFAULT, RETURNS THE FIRST OCCURRENCE. THUS, FOR THOSE FIELDS WHICH HAVE ONLY ONE OCCURRENCE, THE USER NEED NEVER SPECIFY ANYTHING MORE THAN THE FIELD NAME - 'ADR', 'NAMID', ETC.

WE COULD CHANGE THE THIRD CLASS IN THE ABOVE LIST FROM 'BIOLOGY8' TO 'MATH7' BY USING THE 'PT' FUNCTION:

```
          'MATH7' PT CLASS,3   .
```

THIS WILL PUT THE VALUE 'MATH7' IN THE FIELD 'CLASS,3'.

WE CAN ALSO USE THE 'PT' FUNCTION TO CREATE A NEW OCCURRENCE OF THE CLASS FIELD:

```
          'PHYSICS2' PT CLASS,5   .
```

SINCE THE FIELD 'CLASS,5' DOES NOT EXIST BEFORE WE EXECUTE THIS 'PT', THE FUNCTION CREATES THE FIELD AND INSERTS THE VALUE 'PHYSICS2'. THIS ABILITY TO CREATE FIELDS IS NOT LIMITED TO THOSE FIELDS ALREADY HAVING SEVERAL OCCURRENCES. FOR EXAMPLE, IF THERE WERE NO 'CLASS' FIELDS PRESENT WE COULD CREATE THE FIRST ONE BY TYPING:

```
            'MATH1' PT CLASS
```

OR

```
            'MATH1' PT CLASS,1   .
```

SIMILARLY, IF THE RECORD CONTAINED NO FIELD WITH THE PHONE NUMBER, WE COULD CREATE ONE BY TYPING:

```
            9283787 PT PHONE   .
```

IF A FIELD CONTAINS AN UNKNOWN NUMBER OF OCCURRENCES, WE CAN ADD A NEW OCCURRENCE WITH THE 'AD' FUNCTION:

```
            'PHYSICS2' AD CLASS   .
```

THIS WILL ADD A NEW OCCURRENCE AND INSERT THE VALUE 'PHYSICS2'. IF THE HIGHEST OCCURRENCE BEFORE THE USE OF 'AD' IS N, THEN THE EFFECT OF 'AD' IS IDENTICAL TO:

```
            'PHYSICS2' PT CLASS,N+1   .
```

NOW, LETS WRITE A FUNCTION TO LIST THE NAMES OF ALL PEOPLE WHO MEET SOME CRITERIA 'X':

```
            ∇ LISTNAMES X
            [1] LI:  GT NAME
            [2]  → NEXT/LI ∇   .
```

SUPPOSE WE WANT TO USE THIS FUNCTION TO LIST THE NAMES OF ALL PEOPLE TAKING CLASS 'MATH1':

```
            LISTNAMES CLASS IS 'MATH1'
        GEORGE ALLEN GREEN
        WILLIAM ROBERT JONES
        SANDRA ELLEN MACBETH   .
```

THE FIRST LINE OF 'LISTNAMES' SIMPLY GETS THE STUDENT'S NAME AND PRINTS IT OUT. THE SECOND LINE TRIES TO RETRIEVE THE NEXT RECORD SATISFYING THE RETRIEVAL CONDITION AND BRANCHES BACK TO THE FIRST LINE IF ONE IS FOUND (THE FUNCTION 'NEXT' RETRIEVES THE NEXT RECORD MATCHING THE CONDITIONS LAST SETUP BY 'IS' AND RETURNS A '1' IF SUCH A RECORD IS FOUND AND A '0' IF IT IS NOT). IF NO RECORDS ARE FOUND THAT SATISFY THE CRITERIA (I.E. NO ONE IS TAKING CLASS 'MATH1'), THE FUNCTION 'GT' IN LINE ONE OF 'LISTNAMES' WILL RETURN ι0, AND LINE TWO WILL FALL THROUGH AND TERMINATE EXECUTION.

> WHEN THE FUNCTION 'LISTNAMES' IS FIRST ENTERED, 'X' CONTAINS '1'. THIS '1' IS THE RESULT OF EXECUTING 'CLASS IS 'MATH1'' AND SIGNIFIES THAT THE SYSTEM HAS FOUND THE FIRST RECORD CONTAINING THE VALUE 'MATH1'. 'X', THE RIGHT ARGUMENT OF 'LISTNAMES', IS ONLY USED AS A CONVENIENT WAY TO HANG THE RETRIEVAL CONDITIONS ON THE SAME LINE AS THE FUNCTION CALL. 'X' HAS NO FURTHER USE INSIDE THE FUNCTION.

CONSIDER THE DEFINITIONS OF SOME OF THE FIELDS USED ABOVE. WE COULD DEFINE THE 'CLASS' FIELD AS:

```
CLASS ← 5 FIELD CHR 12 INDEX 200  .
```

THIS DEFINES THE FIELD AS 1) *HAVING THE FIELD NUMBER* 5, 2) *AS BEING A CHARACTER FIELD OF WIDTH* 12, *AND* 3) *AS HAVING AN INDEX CONTAINING AN ESTIMATED* 200 *UNIQUE VALUES (THIS ESTIMATE COULD BE WILDLY WRONG. BAD ESTIMATES WILL NEVER PRODUCE ERRORS IN THE SYSTEM. THEY MAY, HOWEVER, SLOW DOWN EXECUTION OR CAUSE THE INDEX TO CONSUME MORE SPACE THAN NEEDED.)*

AS ANOTHER EXAMPLE OF FIELD DEFINITION, WE COULD DEFINE THE 'AVG' FIELD (CONTAINING THE STUDENT'S MARK AVERAGE) AS:

```
AVG ← 7 FIELD NUM 1 SCALE 1  .
```

THIS ASSIGNS THE FIELD-NUMBER 7 *TO THE 'AVG' FIELD AND DEFINES THE FIELD AS BEING NUMERIC WITH ONE ELEMENT AND ONE POSITION TO THE RIGHT OF THE DECIMAL (I.E.* 3.7).

AS A FINAL EXAMPLE, WE COULD DEFINE THE 'ADR' FIELD AS:

```
ADR ← 3 FIELD CHR 0  .
```

THIS ASSIGNS THE FIELD NUMBER 3 *TO THE 'ADR' FIELD AND DEFINES THE FIELD AS A VARIABLE-LENGTH CHARACTER VECTOR. (A FIELD WIDTH OF* 0 *CAUSES THE FIELD TO BE VARIABLE-LENGTH.)*

DESCRIPTION OF THE SYSTEM

SYSTEM STEWARD VS. SYSTEM USERS

A PRIMARY DISTINCTION IN THE BROWNBOX SYSTEM IS THAT BETWEEN THE SYSTEM STEWARD AND NORMAL SYSTEM USERS. THE STEWARD HAS THE RESPONSIBILITY OF SETTING THE SYSTEM UP AND MAINTAINING ITS INTEGRITY. HE DEFINES THE TYPES OF FIELDS THAT CAN BE USED AND ASSIGNS MNEMONICS OR FIELD NAMES TO THESE FIELDS. HE ALSO CONTROLS THE MANNER IN WHICH USERS CAN ACCESS THE FILES, GIVING DIFFERENT ACCESS PERMISSIONS TO DIFFERENT USERS. IT IS THE STEWARDS'S RESPONSIBILITY TO KEEP THE FILE STRUCTURE UP-TO-DATE: TO ADD NEW FIELDS, REMOVE OLD FIELDS, CHANGE ACCESS PERMISSIONS; TO INSERT OR DELETE INDEXES OVER FIELDS AS THE NEED ARISES OR DECLINES.

THE USERS, ON THE OTHER HAND, ARE CONCERNED WITH THE INFORMATION STORED WITHIN THE FILE, NOT WITH THE FILE STRUCTURE ITSELF. USERS CREATE NEW RECORDS, DESTROY OLD RECORDS, RETRIEVE RECORDS, AND ALTER RECORDS. THE USER ENTERS NEW INFORMATION INTO THE FILE AND RETRIEVES INFORMATION ALREADY PRESENT. THERE MIGHT, IN FACT, BE SEVERAL DIFFERENT LEVELS OF USERS. THE HIGHEST LEVEL WOULD DESIGN ROUTINES AND MECHANISMS TO BE USED BY LOWER LEVELS. THESE ROUTINES WOULD CONSTITUTE A SORT OF LANGUAGE TAILORED TO THE PARTICULAR APPLICATIONS AT HAND. THIS LANGUAGE WOULD PROVIDE CONVENIENT MECHANISMS FOR INPUTTING DATA, FORMATTING OUTPUT, AND REQUESTING VARIOUS TYPES OF INFORMATION. THE LANGUAGE WOULD, IN FACT, ACT AS AN INTERFACE BETWEEN THE LOW-LEVEL USER AND THE BROWNBOX PROPER. THE LOWEST LEVEL USER WOULD SEE VERY LITTLE OF THE BROWNBOX ITSELF. THERE IS, IN FACT, NO REASON WHY THE ULTIMATE USER NEED KNOW ANY APL AT ALL.

FIELD DEFINITION AND ACCESS

FIELDS MUST BE DEFINED BY THE STEWARD BEFORE ANYONE CAN USE THEM. THE FIELD DEFINITION CONSISTS OF:

1) *THE FIELD NUMBER* (1-10,000)
2) *THE MNEMONIC USED TO REPRESENT THE FIELD NUMBER*
3) *THE FIELD TYPE - CHARACTER, NUMERIC, OR POINTER (TO BE EXPLAINED LATER)*
4) *THE FIELD WIDTH* (1-150 *FOR FIXED-LENGTH FIELDS,* 0 *FOR VARIABLE-LENGTH FIELDS*)
5) *THE SCALE FACTOR FOR NUMERIC FIELDS - I.E. THE NUMBER OF SIGNIFICANT DIGITS TO BE PRESERVED TO THE RIGHT OF THE DECIMAL POINT.*
6) *THE INDEX SIZE, IF THERE IS TO BE AN INDEX OVER THE FIELD.*

THE FACT THAT A FIELD IS DEFINED DOESN'T MEAN THAT IT EVER HAS TO BE USED. IT MIGHT NOT EVER OCCUR IN ANY RECORD OF THE FILE. DEFINITION JUST OPENS THE DOOR OF POSSIBILITY: ONCE DEFINED A FIELD CAN BE USED WHENEVER NEEDED, BUT UNDEFINED IT CAN'T BE USED AT ALL. THE SET OF DEFINITIONS IS NOT NECESSARILY FIXED AT THE TIME THE SYSTEM IS INITIALIZED. NEW DEFINITIONS CAN BE ADDED LATER AND OLD ONES DELETED. AT THE TIME WHEN AN OLD FIELD DEFINITION IS DELETED, ALL EXISTING OCCURRENCES OF THAT FIELD IN THE FILE ARE ALSO REMOVED.

THE STEWARD CONTROLS ACCESS TO THE FILE ON A FIELD-BY-FIELD AND USER-BY-USER BASIS. THE USER CAN BE GIVEN ANY COMBINATION OF READ AND/OR WRITE ACCESS TO EACH FIELD. HE COULD HAVE READ-ONLY ACCESS TO ONE SET OF FIELDS, READ/WRITE ACCESS TO A SECOND SET, WRITE-ONLY TO A THIRD SET, AND NO ACCESS TO THE REMAINDER. THE STEWARD, IN ADDITION TO CONTROLLING THE USER'S ACCESS TO FIELDS, ALSO CONTROLS THE USER'S ACCESS TO THE RECORD CREATE-DESTROY MECHANISM: THE USER CAN BE GIVEN PERMISSION TO CREATE NEW RECORDS AND/OR DESTORY OLD RECORDS.

DIFFERENT USERS WOULD BE GIVEN DIFFERENT LEVELS OF ACCESS. THE STEWARD WOULD NORMALLY HAVE COMPLETE ACCESS. SOME PRIVELEGED USERS WOULD PROBABLY HAVE READ ACCESS TO ALL FIELDS AND WRITE ACCESS TO MOST. OTHERS WOULD HAVE READ ACCESS TO EVERYTHING BUT NO WRITE ACCESS. STILL OTHERS WOULD ONLY HAVE READ ACCESS TO A LIMITED SET OF FIELDS. OTHER APL USERS (I.E. THOSE WHO HAD NO

CONNECTION WITH THIS PARTICULAR BROWNBOX SYSTEM) WOULD HAVE NO ACCESS WHATSOEVER. (LIKEWISE, USERS OF OTHER BROWNBOX SYSTEMS WOULD HAVE ABSOLUTELY NO ACCESS TO THIS BROWNBOX SYSTEM).

STRUCTURE OF THE FILE

THE MAIN FILE IS A SET OF RECORDS RUNNING FROM RECORD NUMBER ONE TO THE LAST RECORD IN THE FILE. ALTHOUGH THE LOW-LEVEL STRUCTURE OF THE FILE IS, IN FACT, SEQUENTIAL, THE USER WILL RARELY, IF EVER, TREAT THE FILE AS A SEQUENCE OF RECORDS. HE WILL, INSTEAD, TREAT IT AS A COMBINATION OF STRUCTURES: ASSOCIATIVE, CHAINED, AND LINKED.

THE FILE WILL BE ASSOCIATIVE WHEN THE USER IS TRYING TO RETRIEVE RECORDS BY SPECIFYING THE CONTENTS OF PARTICULAR FIELDS. FOR EXAMPLE, THE USER MIGHT SPECIFY THAT THE 'NAMID' FIELD CONTAINS 'JS GREEN' OR THAT SOME 'CLASS' FIELD CONTAINS 'MATH7'. IN ORDER TO IMPLEMENT THE SYSTEM EFFICIENTLY (WITH HASH CODING), THE NATURE OF THIS ASSOCIATIVITY IS QUITE RESTRICTED. RECORDS CAN BE RETRIEVED ONLY ON EXACT MATCHES TO THE CONTENTS OF SINGLE FIELDS. THERE IS NO ELEMENTARY CAPABILITY FOR FINDING RECORDS WHICH MEET A RANGE OF CRITERIA. FOR EXAMPLE, THERE IS NO BASIC OPERATION THAT WILL FIND ALL STUDENTS WHOSE MARK AVERAGE IS GREATER THAN 3.5. NOR IS THERE A BASIC OPERATION THAT WILL FIND THOSE STUDENTS WHO ARE TAKING BOTH 'MATH6' AND 'PHYSICS3'. ALL THIS IS NOT TO SAY THAT THESE MORE COMPLICATED RETRIEVALS ARE IMPOSSIBLE - THEY CAN BE PROGRAMMED IN IF NECESSARY.

> FOR EXAMPLE, WE COULD GET STUDENT'S WHOSE MARKS WERE WITHIN A PARTICULAR RANGE BY STORING A NEW FIELD - 'MARKRANGE'. 'MARKRANGE' WOULD CONTAIN THE STUDENT'S MARK ROUNDED DOWN TO THE NEXT MULTIPLE OF 0.5. THAT IS, 'MARKRANGE' IS $(\lfloor 2\times AVG)\div 2$. THUS, ALL STUDENTS WHOSE MARK AVERAGE WAS GREATER THAN OR EQUAL TO 3.5 AND LESS THAN 4.0 WOULD HAVE 'MARKRANGE' EQUAL TO 3.5. ALL STUDENTS WITHIN A MARK RANGE COULD NOW BE RETRIEVED USING THE ELEMENTARY OPERATION 'IS' - 'MARKRANGE IS 3.5'.
>
> AS A FURTHER EXAMPLE, WE COULD FIND THOSE STUDENTS TAKING BOTH 'MATH6' AND 'PHYSICS3' SIMPLY BY RETRIEVING ALL STUDENTS TAKING 'MATH6' AND REJECTING THOSE WHO ARE NOT ALSO TAKING 'PHYSICS3'. IT MUST BE OBSERVED, HOWEVER, THAT THIS PROCESS COULD GET FAIRLY INEFFICIENT IF ONLY A SMALL PERCENTAGE OF STUDENTS WHO TAKE 'MATH6' ALSO TAKE 'PHYSICS3'.

IN ADDITION TO THE ASSOCIATIVE STRUCTURE DISCUSSED ABOVE, THE FILE MIGHT ALSO HAVE A CHAINED STRUCTURE. THIS WOULD APPEAR WHEN THE USER IS RETRIEVING THE ENTIRE SET OF RECORDS THAT MATCH SOME CRITERION (FOR EXAMPLE, ALL STUDENTS TAKING 'MATH7'). THE FIRST RECORD WILL BE CHAINED TO THE SECOND, THE SECOND TO THE THIRD, ETC. (THE USER CAN TRACE DOWN THIS CHAIN BY USING THE 'NEXT' FUNCTION). CHAINS ARE MAINTAINED IN INVERSE CHRONOLOGICAL ORDER - THE OLDEST ITEM WILL BE AT THE END OF THE CHAIN AND THE YOUNGEST ITEM WILL BE AT THE FRONT. THAT IS, WHEN TRACING DOWN A CHAIN, THE LAST ITEM TO BE ADDED TO THE FILE WILL BE THE FIRST ONE RETRIEVED, AND THE FIRST ITEM ADDED WILL BE THE LAST ONE RETRIEVED.

FILES MIGHT ALSO HAVE A LINKED STRUCTURE WHERE SOME RECORDS CONTAIN POINTERS TO OTHER RECORDS. FOR EXAMPLE, THE MAIN RECORD FOR A STUDENT MIGHT CONTAIN A POINTER TO A SUBSIDIARY HISTORICAL RECORD CONTAINING INFORMATION FROM PREVIOUS TERMS - CLASSES TAKEN, INSTRUCTORS, GRADES RECEIVED, ETC. ONCE THE MAIN RECORD IS IN MEMORY, THE POINTER ALLOWS EFFICIENT RETRIEVAL OF THE HISTORICAL RECORD. MAINTAINING SETS OF LINKED RECORDS ALLOWS THE TOTAL INFORMATION STORED ON ANY ITEM IN THE FILE TO GROW ARBITRARILY. AT THE SAME TIME, HOWEVER, THE SIZE OF INDIVIDUAL RECORDS IS RESTRICTED SO THAT THEY COMFORTABLY FIT WITHIN AN ACTIVE WORKSPACE.

ANOTHER ADVANTAGE OF LINKED RECORDS IS THAT THEY ALLOW ARBITRARILY COMPLICATED DATA STRUCTURES - RECORDS CAN POINT TO RECORDS THAT POINT TO RECORDS, ETC. FOR EXAMPLE, WE COULD CONSTRUCT A HIERARCHIAL OR TREE STRUCTURE WHERE A MASTER RECORD AT THE TOP HAS A SET OF SUBORDINATE RECORDS EACH OF WHICH, IN TURN, HAS ITS OWN SUBORDINATE RECORDS. OR WE COULD HAVE A STRUCTURE WHERE SETS OF RECORDS SHARE COMMON SUBRECORDS. THIS WOULD ALLOW THE CONCENTRATION OF RELEVANT INFORMATION IN A FEW WELL-DEFINED SPOTS AND PREVENT DUPLICATION AND THE CONSEQUENT WASTE OF STORAGE.

ALL LINKAGES ARE TWO-WAY, IF THE USER LINKS RECORD A TO RECORD B, THE SYSTEM AUTOMATICALLY INSERTS AN INVERSE LINK FROM B TO A. FOR EXAMPLE, WHEN THE USER FIRST INSERTS A POINTER TO THE HISTORICAL RECORD INTO A STUDENT'S MAIN RECORD, THE SYSTEM WILL IMMEDIATELY INSERT A BACKPOINTER TO THE MAIN RECORD INTO THE HISTORICAL RECORD. SIMILARLY, IF THE USER REMOVES THE FORWARD POINTER, THE SYSTEM WILL AUTOMATICALLY REMOVE THE BACKWARD POINTER. THIS PROCESS ALSO WORKS IN REVERSE - IF THE USER INSERTS (OR DELETES) A BACKWARD POINTER, THE SYSTEM WILL INSERT (OR DELETE) THE FORWARD POINTER. THIS SYMMETRICAL MAINTENANCE OF ALL LINKS INSURES THAT THE FILE CONTAINS NO ORPHAN RECORDS WHOSE

PARENTS CANNOT BE FOUND. GIVEN ANY RECORD WE CAN ALWAYS FIND ALL RECORDS LINKED TO IT, HOWEVER INDIRECT THESE LINKS MAY BE.

ALL OF THE ABOVE STRUCTURES - ASSOCIATIVE, CHAINED, AND LINKED - ARE SUPERIMPOSED OVER THE BASICALLY SEQUENTIAL SUBSTRUCTURE OF THE FILE. IN THIS SUBSTRUCTURE EACH RECORD OCCUPIES ONE FILE POSITION - FROM POSITION 1 TO POSITION N. A RECORD IS UNIQUELY IDENTIFIED BY ITS POSITION IN THE FILE. WE REFER TO THIS POSITION AS THE GROUP OR RECORD ID. FOR EXAMPLE, THE RECORD WITH GROUP ID 3 IS THE RECORD THAT OCCUPIES THE THIRD POSITION IN THE SEQUENTIAL FILE.

NOT ALL POSITIONS IN THE FILE ARE OCCUPIED. WHEN A RECORD IS DESTROYED, ITS POSITION IN THE FILE IS LEFT VACANT. WHEN NEW RECORDS ARE CREATED, THEY REOCCUPY THE VACANT FILE POSITIONS. IF THERE ARE NO VACANT POSITIONS, NEW RECORDS ARE APPENDED TO THE END OF THE FILE. ONCE A FILE GROWS TO FILL N POSITIONS, IT WILL NEVER SHRINK TO FEWER POSITIONS. THE SIZE OF THE FILE (NUMBER OF BYTES USED) MAY DECREASE WHEN RECORDS ARE DESTROYED, BUT THE LENGTH OF THE FILE WILL REMAIN THE SAME.

> THE REASON THAT THE LENGTH OF THE FILE IS NOT SHRUNK WHEN RECORDS ARE DESTROYED IS THAT SUCH AN OPERATION WOULD BE EXCEEDINGLY TIME CONSUMING. ALL RECORDS FOLLOWING THE DELETED RECORD WOULD HAVE TO BE MOVED BACKWARDS, AND ALL LINKS TO THESE RECORDS WOULD HAVE TO BE ADJUSTED.
>
> IT WOULD BE POSSIBLE TO WRITE A FILE COMPRESSION ROUTINE, BUT THIS HAS NOT BEEN DONE. THIS WOULD ONLY BE ADVANTAGEOUS FOR THOSE FILES WHICH GROW VERY LARGE AT SOME PERIOD OF TIME AND SHRINK DRASTICALLY THEREAFTER. THE GENERAL ASSUMPTION THAT HAS BEEN MADE IS THAT MOST FILES WILL HAVE A TENDENCY TO CONTINUALLY GROW UNTIL THEY ARE NO LONGER NEEDED AND ARE DESTROYED IN THEIR ENTIRETY.

THE ONLY TIME THAT THE USER WOULD ACTUALLY WORK WITH THE SEQUENTIAL FILE WOULD BE WHEN HE WAS SCANNING THE WHOLE FILE - EITHER TO ACCUMULATE TOTALS OR TO SEARCH FOR SOME ITEM NOT IN AN INDEX. THE SCAN WOULD PROCEED BY FETCHING RECORD 1, THEN RECORD 2, RECORD 3, ON UP TO RECORD N, WHERE N IS THE LAST RECORD IN THE FILE. 'FETCH I' WILL FETCH THE I'TH RECORD AND RETURN A 1 IF THE RECORD IS FOUND AND A 0 IF IT IS NOT FOUND - 0 WILL BE RETURNED FOR ALL DESTROYED RECORDS. THIS PROCEDURE WILL RETRIEVE ALL RECORDS IN THE FILE. HOWEVER, SINCE THE USER MAY NOT, IN FACT, WANT ALL RECORDS, HE WILL HAVE TO REJECT THOSE RECORDS WHICH ARE NOT OF THE CORRECT TYPE. FOR EXAMPLE, IF WE WERE SCANNING A

STUDENT FILE IN ORDER TO COLLECT STATISTICS ON THE DISTRIBUTION OF MARK AVERAGES, WE WOULD REJECT ALL RECORDS NOT CONTAINING AN 'AVG' FIELD. THAT IS, WE WOULD REJECT THE RECORDS CONTAINING HISTORICAL INFORMATION ON STUDENTS AND RECORDS CONTAINING DESCRIPTIVE INFORMATION ABOUT COURSES.

STRUCTURE OF A RECORD

A RECORD IS INTERNALLY STRUCTURED AS A SEQUENCE OF FIELDS. EACH FIELD CONSISTS OF A FIELD IDENTIFIER FOLLOWED BY THE FIELD CONTENTS. THE IDENTIFIER CONSISTS OF THE FIELD NUMBER AND THE OCCURRENCE NUMBER:

```
        (NAME) JOHN SMITH  (ADR) 703 WILCOX RD.;TORONTO
             (CLASS,1) MATH1  (NAMID) JK SMITH
             (CLASS,2) PHYSICS3  (PHONE) 7335656  (CLASS,3) CHEM4
```

NOTICE THAT WE FOLLOW THE CONVENTION OF DROPPING THE OCCURRENCE NUMBER FOR THOSE FIELDS WHICH ARE SUPPOSED TO OCCUR ONLY ONCE, SUCH AS THE 'NAME' FIELD, 'ADR' FIELD, ETC. NOTE ALSO THAT ALL THE OCCURRENCES OF A FIELD NEED NOT LIE ADJACENT TO EACH OTHER - THE THREE OCCURRENCES OF THE CLASS FIELD ARE SCATTERED IN DIFFERENT PARTS OF THE RECORD.

FOR THE SAKE OF FURTHER DISCOURSE, WE WILL CONSTRUCT THE FOLLOWING ABSTRACT RECORD:

```
R1:     (F1)XA  (F2,1)XB  (F3)XC  (F2,2)XD
```

HERE 'F1' THROUGH 'F3' REPRESENT FIELD NUMBERS IN THE SAME SENSE THAT THE MNEMONICS 'CLASS', 'NAME', 'ADR', ETC. REPRESENT FIELD NUMBERS. 'XA' THROUGH 'XD' REPRESENT THE VALUES CONTAINED IN THESE FIELDS.

WHEN WE ADD A NEW FIELD OR A NEW OCCURRENCE OF AN EXISTING FIELD, WE APPEND THIS NEW VALUE ONTO THE END OF THE RECORD. FOR EXAMPLE:

```
        XE AD F2
```

WOULD CHANGE THE ABOVE RECORD TO:

```
        (F1)XA  (F2,1)XB  (F3)XC  (F2,2)XD  (F2,3)XE .
```

WHEN WE CHANGE THE VALUE OF AN EXISTING FIELD OR OCCURRENCE, THE NEW VALUE REPLACES THE OLD VALUE WITHOUT ALTERING THE SEQUENCE OF FIELDS. FOR EXAMPLE,

```
XF PT F3
```

WOULD CHANGE THE ABOVE R1 RECORD TO:

```
(F1)XA  (F2,1)XB  (F3)XF  (F2,2)XD  .
```

DELETING A FIELD CAUSES IT TO COMPLETELY DISAPPEAR FROM THE SEQUENCE. THE RESULT IS INDISTINGUISHABLE FROM THE CASE WHERE THE FIELD HAS NEVER BEEN INSERTED IN THE FIRST PLACE.

```
''PT F2,1
```

CHANGES THE R1 RECORD TO:

```
(F1)XA  (F3)XF  (F2,2)XD  .
```

NOTICE THAT THIS DELETION DOES NOT CAUSE THE OLD SECOND OCCURRENCE OF 'F2' TO BECOME THE NEW FIRST OCCURRENCE. OCCURRENCE NUMBERS ARE ABSOLUTE - IT IS PERFECTLY POSSIBLE TO INSERT THE N'TH OCCURRENCE WITHOUT HAVING INSERTED ALL (OR ANY) OF THE OCCURRENCES BEFORE N.

```
XG PT F2,333
```

CHANGES THE R1 RECORD TO:

```
(F1)XA  (F2,1)XB  (F3)XC  (F2,2)XD  (F2,333)XG  .
```

A COMPLETE LIST OF ALL THE FIELDS IN A RECORD CAN BE GOTTEN WITH THE 'FL' FUNCTION. THIS FUNCTION RETURNS A NUMERIC MATRIX WITH AS MANY ROWS AS THERE ARE FIELDS IN THE ACTIVE RECORD. THE FIRST COLUMN CONTAINS THE FIELD NUMBERS AND THE SECOND COLUMN CONTAINS THE CORRESPONDING OCCURRENCE NUMBERS. FOR THE R1 RECORD,

```
FL
```

WOULD RESULT IN:

```
1  1
2  1
3  1
2  2  .
```

WHERE 1, 2, AND 3 ARE, RESPECTIVELY, THE FIELD NUMBERS FOR 'F1', 'F2', AND 'F3'.

ONLY THOSE FIELDS THAT THE USER HAS READ ACCESS TO ARE VISIBLE TO HIM. FIELDS HE DOES NOT HAVE ACCESS TO WILL BE COMPLETELY INVISIBLE - NOT ONLY WILL THE USER BE UNABLE TO GET THE CONTENTS OF THESE FIELDS OR TO SEARCH THEM FOR PARTICULAR VALUES, BUT HE WILL EVEN BE UNABLE TO DETERMINE WHETHER OR NOT THE FIELDS ARE PRESENT IN THE RECORD AT ALL. FOR EXAMPLE, SUPPOSE THAT THE USER DID NOT HAVE READ ACCESS TO THE FIELD 'F2' IN THE RECORD R1. THEN,

```
        FL
```

WOULD RETURN

```
        1 1
        3 1 ,
```

OMITTING ALL REFERENCE TO FIELD NUMBER 2. ALL OCCURRENCES OF FIELD 'F2', HOWEVER, WOULD STILL BE PRESENT IN THE RECORD AND WOULD BE VISIBLE TO ANYONE HAVING THE APPROPRIATE ACCESS.

NORMALLY, FIELD NUMBERS LIE IN THE RANGE 1 TO 10,000, BUT THERE IS ONE EXCEPTION. NEGATIVE FIELDS ARE USED TO IDENTIFY THOSE FIELD NUMBERS CONTAINING INVERSE POINTERS - I.E. BACKWARD LINKS. IF FORWARD LINKS OF A PARTICULAR TYPE ARE STORED IN FIELD NUMBER N, THEN BACKWARD LINKS OF THIS TYPE ARE STORED IN FIELD NUMBER ¯N. THUS, IF A MAIN STUDENT RECORD HAS A POINTER TO A HISTORICAL RECORD STORED IN FIELD NUMBER 8, THEN THE HISTORICAL RECORD HAS A BACKPOINTER TO THE MAIN RECORD STORED IN FIELD NUMBER ¯8. THESE NEGATIVE FIELD NUMBERS WILL SHOW UP IN THE LIST OF FIELD NUMBERS PRODUCED BY 'FL'.

> *THERE IS, IN FACT, A SECOND KIND OF NEGATIVE FIELD USED IN THE SYSTEM, BUT THESE ALWAYS REMAIN COMPLETELY INVISIBLE TO ALL USERS. THEY ARE USED IN THE MAINTENANCE OF INDEXES FOR THOSE FIELDS WHICH HAVE INDEXES. THUS, FIELD NUMBER 5 IN THE STUDENT SYSTEM (THE 'CLASS' FIELD) HAS AN INDEX AND, THEREFORE, FOR EVERY OCCURRENCE OF FIELD 5 THERE IS A CORRESPONDING OCCURRENCE OF FIELD ¯5. THE ONLY EFFECT OF ALL THIS THAT THE USER MIGHT NOTICE IS THAT THE NEGATIVE INDEX FIELDS WILL TAKE UP SPACE IN THE RECORD. THUS, RECORDS WITH A LOT OF INDEXED FIELDS COULD CONSUME MORE MEMORY SPACE THAN THE USER MIGHT OTHERWISE SUSPECT.*

SNAPSHOTS OF THE FILE

WHEN A USER IS WORKING WITH THE FILE, HE WORKS WITH A SNAPSHOT OF THE FILE. CONCEPTUALLY, A SNAPSHOT IS EQUIVALENT TO A COMPLETE COPY OF THE FILE TAKEN INSTANTANEOUSLY. SNAPSHOTS ARE USED IN ORDER TO INSURE THAT SEVERAL USERS SIMULTANEOUSLY ACCESSING THE FILE DO NOT INTERFERE WITH EACH OTHER. IN PARTICULAR, WE WANT TO MAKE SURE THAT SOMEONE WHO IS WRITING ON THE FILE DOES NOT AFFECT OTHER USERS WHO ARE READING (SINCE THE SYSTEM ALLOWS ONLY ONE WRITE AT A TIME, WE DON'T HAVE TO WORRY ABOUT WRITES INTERFERING WITH WRITES).

CONSIDER THE FOLLOWING: AT 2:00 USER X STARTS READING THE FILE IN ORDER TO ACCUMULATE VARIOUS STATISTICS AND TOTALS. THIS IS A LONG PROCESS, AND IT WILL TAKE UNTIL 3:00 TO COMPLETE. MEANWHILE, AT 2:30, USER Y STARTS UPDATING VARIOUS RECORDS IN THE FILE. IF Y UPDATES THE SAME RECORDS THAT X IS READING, X MAY GET INVALID RESULTS. FOR EXAMPLE, SUPPOSE THAT X IS TRYING TO FIND OUT WHICH STUDENTS HAVE MARK AVERAGES OF EITHER 3.9 OR 4.0. IF THE 'AVG' FIELD HAS AN INDEX OVER IT, THIS INDEX CAN BE USED FIRST TO RETRIEVE ALL STUDENTS WITH 'AVG' OF 3.9, AND THEN TO RETRIEVE ALL STUDENTS WITH 'AVG' OF 4.0. LETS SUPPOSE THAT X HAS JUST FINISHED RETRIEVING THE 3.9'S AND IS JUST STARTING ON THE 4.0'S. IF, AT THIS TIME, Y STARTS UPDATING MARKS, SOME STUDENTS WHO PREVIOUSLY HAD 4.0 COULD FALL BACK INTO THE 3.9 CATEGORY. IF THESE 4.0 STUDENTS CHANGE TO 3.9 BEFORE X GETS TO THEM, X WILL COMPLETELY MISS THEM, EVEN THOUGH THEY STILL LIE WITHIN THE 3.9 TO 4.0 RANGE.

IN GENERAL, IF USER Y UPDATES THE FILE WHILE USER X IS READING, X WILL NOT BE ABLE TO GET A COHERENT SET OF DATA. RECORDS WILL TAKE ON AN EPHEMERAL QUALITY - THEY WILL CHANGE FROM ONE READING TO THE NEXT. THIS WILL CAUSE INCONSISTENCIES - SOME RECORDS WILL BE MISSED AND OTHERS WILL BE PROCESSED TWICE. SNAPSHOTS COMPLETELY CURE THIS PROBLEM - IN THE ABOVE CASE USER X WILL WORK WITH A SNAPSHOT OF THE FILE TAKEN AT 2:00. CHANGES MADE BY Y WILL NOT BE SEEN BY X. X WILL SEE ONLY THE ORIGINAL FILE UP UNTIL SUCH A TIME THAT HE DECIDES TO TAKE A NEW SNAPSHOT.

THE SYSTEM WILL SUPPORT AN ARBITRARY NUMBER OF SIMULTANEOUS SNAPSHOTS. USER W COULD TAKE A SNAPSHOT AT 1:00, USER X COULD TAKE ONE AT 2:05, USER Y AT 2:10, USER Z AT 2:15, ETC. AT 2:30 ALL OF THESE USERS COULD LOOK AT RECORD 736 OF THE FILE, AND EACH OF THEM MIGHT SEE SOMETHING DIFFERENT. THIS WOULD OCCUR, FOR EXAMPLE, IF RECORD 736 WERE VERY ACTIVE AND HAD BEEN UPDATED AT 2:01, 2:06, 2:11, 2:16, ETC.

WITH THE SNAPSHOT MECHANISM, THE USER CAN KEEP HIS INFORMATION AS COHERENT OR AS UP-TO-DATE AS HE WANTS. IF HE WANTS COHERENT DATA, HE TAKES ONE SNAPSHOT AND DOES ALL OF HIS WORK WITH IT. IF HE WANTS UP-TO-DATE INFORMATION, HE TAKES MANY SNAPSHOTS, TAKING A NEW ONE WHENEVER THE OLD ONE HAS BEEN AROUND LONG ENOUGH TO GET STALE. (A SNAPSHOT IS TAKEN WHENEVER THE USER CALLS THE FUNCTION 'READ'. THIS FUNCTION DECLARES TO THE SYSTEM THAT THE USER WANTS TO GO INTO A READ-STATE, WHEREUPON THE SYSTEM SUPPLIES THE USER WITH A SNAPSHOT.)

IT OUGHT TO BE EMPHASIZED HERE THAT A SNAPSHOT ONLY APPEARS TO BE A COMPLETE COPY OF THE FILE. NO PHYSICAL COPY IS, IN FACT, MADE. THE SYSTEM ACHIEVES THE EFFECT OF SNAPSHOTS BY MONITORING UPDATED INFORMATION AND BY MAINTAINING SMALL ARCHIVAL FILES. THESE SMALL ARCHIVES CONTAIN THE DIFFERENT GENERATIONS OF DATA THAT ARE NEEDED BY DIFFERENT SNAPSHOTS.

SNAPSHOTS ARE ALSO USED WHEN THE USER IS WRITING ON THE FILE. HERE THE USER CREATES A WRITE SNAPSHOT BY CALLING THE 'WRITE' FUNCTION. THIS FUNCTION REQUESTS THE SYSTEM TO PUT THE USER INTO WRITE-STATE. ONCE THE USER HAS HIS WRITE SNAPSHOT, HE CAN PROCEED TO MODIFY THE FILE. IN FACT, HOWEVER, HE WILL BE MODIFYING HIS SNAPSHOT. WHEN HE IS DONE MODIFYING, THE SNAPSHOT CAN BE MERGED WITH THE ACTUAL FILE. THIS IS DONE EITHER BY CALLING THE 'DONE' FUNCTION OR BY REQUESTING A NEW READ OR WRITE SNAPSHOT. NEW SNAPSHOTS AUTOMATICALLY ISSUE A 'DONE' FOR ANY OLD SNAPSHOT STILL IN EFFECT. UNTIL 'DONE' IS CALLED, MODIFICATIONS WILL EXIST ONLY ON THE SNAPSHOT AND NOT ON THE FILE ITSELF. THUS, UNTIL ALL MODIFICATIONS ARE COMPLETED, NO OTHER USER WILL BE ABLE TO SEE ANY CHANGE TO THE FILE AT ALL. THAT IS, IF ONE USER IS MAKING CHANGES TO A SET OF RECORDS, OTHER USER'S SNAPSHOTS WILL PICTURE THE FILE EITHER BEFORE ANY CHANGES ARE MADE OR AFTER ALL CHANGES ARE MADE. NO SNAPSHOT WILL PICTURE SOME OF THE CHANGES BUT NOT OTHERS. THIS INSURES THAT ALL USERS WILL GET COHERENT DATA NO MATTER WHEN THEY ISSUE THEIR SNAPSHOTS.

ANYTIME THAT A USER IS WRITING HE CAN DECIDE NOT TO MERGE HIS CURRENT WRITE SNAPSHOT BACK INTO THE FILE. THIS IS DONE BY CALLING THE FUNCTION 'OUT', WHICH DISCARDS THE CURRENT SNAPSHOT. THIS LEAVES THE FILE IN EXACTLY THAT STATE IT WAS IN WHEN THE WRITE SNAPSHOT WAS ORIGINALLY TAKEN. 'OUT' IS A CONVENIENT WAY TO CORRECT MISTAKES OR TO RESTORE THE FILE AFTER RUNNING TESTS WHICH MAKE CHANGES.

ONCE AGAIN, WE OUGHT TO EMPHASIZE THAT WE HAVE BEEN DESCRIBING THE APPEARANCE OF THE SNAPSHOT MECHANISM AND NOT ITS ACTUAL IMPLEMENTATION. THE APPEARANCE WOULD IMPLY THAT THE 'READ', 'WRITE' AND 'DONE' FUNCTIONS WOULD TAKE A LOT OF TIME, *WHILE THE 'OUT' FUNCTION WOULD TAKE VERY LITTLE. IN FACT, JUST THE OPPOSITE IS TRUE. THE 'READ', 'WRITE', AND 'DONE' FUNCTIONS RUN QUITE QUICKLY WHILE THE 'OUT' FUNCTION TAKES SOME TIME. THIS IS AS IT SHOULD BE, SINCE 'READ', 'WRITE', AND 'DONE' WILL BE USED A LOT MORE OFTEN THAN 'OUT'.*

SYSTEM SECURITY

THE SYSTEM HAS QUITE POWERFUL BUILT-IN SECURITY MECHANISMS. AS MENTIONED BEFORE, A USER'S ACCESS TO FIELDS CAN BE VERY PRECISELY CONTROLLED. FIELDS TO WHICH HE HAS NO ACCESS WILL REMAIN COMPLETELY INVISIBLE. FOR ALL THIS SECURITY TO BE REALLY EFFECTIVE, HOWEVER, WE HAVE TO PREVENT USERS FROM HAVING ACCESS TO INTERNAL FUNCTIONS AND VARIABLES. NORMAL APL PROVIDES NO WAY TO KEEP A USER FROM PLAYING WITH PARTICULAR FUNCTIONS OR GLOBAL VARIABLES. APL⋆PLUS, HOWEVER, DOES HAVE A 'SUPERSEAL' COMMAND THAT MAKES FUNCTIONS AND VARIABLES INVISIBLE. THIS 'SUPERSEAL' CAPABILITY IS NECESSARY IN ORDER TO HIDE FROM THE USER THINGS HE SHOULD NOT SEE. THE ULTIMATE USER OF A BROWNBOX SYSTEM SHOULD PROBABLY SEE ONLY A FEW HIGH-LEVEL FUNCTIONS WHICH INITIATE VARIOUS ACTIONS AFTER ACCEPTING COMMANDS FROM THE KEYBOARD. THAT IS HE WILL PROBABLY INTERACT WITH THE SYSTEM PRIMARILY THROUGH A COMMAND STRUCTURE DESIGNED BY THE STEWARD RATHER THAN THROUGH THE ACTUAL BROWNBOX FUNCTIONS THEMSELVES.

IN ADDITION TO HIDING FUNCTIONS AND VARIABLES FROM THE USER, WE ALSO HAVE TO PREVENT HIM FROM INTERRUPTING FUNCTIONS AND FROM SAVING AND RELOADING WORKSPACES AT ARBITRARY TIMES. WE DO THIS BY SURROUNDING CRUCIAL SECTIONS OF CODE WITH THE 'LOK' AND 'NOK' FUNCTIONS. 'LOK' TURNS THE LOCK MECHANISM ON, AND 'NOK' TURNS IT OFF. WHILE THE LOCK IS ON, THE USER CANNOT GAIN CONTROL OF THE KEYBOARD EXCEPT TO ENTER QUOTE-QUAD INPUT. THE SYSTEM ACCEPTS SUCH INPUT FROM THE USER ONLY WHEN THE 'INPUT' ROUTINE IS CALLED. DIRECT USE OF THE NORMAL APL QUOTE-QUAD MECHANISM WILL CAUSE A LOCK VIOLATION.

THE LOCK MECHANISM WORKS AS FOLLOWS: 'LOK' MEASURES THE ACCUMULATED KEYBOARD UNLOCK TIME (⌶19). WHENEVER THERE IS A FILE OPERATION, THE UNLOCK TIME AT THE TIME OF THE LAST 'LOK' IS COMPARED TO THE CURRENT UNLOCK TIME. IF THERE IS ANY DIFFERENCE, THIS MEANS THAT THE USER HAS ILLEGALLY INTERRUPTED FUNCTION EXECUTION SOMETIME SINCE THE 'LOK'. WHEN SUCH A DIFFERENCE IS DETECTED, THE FILE OPERATION IS NOT DONE AND AN 'OUT' IS EXECUTED INSTEAD. THIS WILL THROW AWAY THE USERS CURRENT SNAPSHOT, LEAVING THE SYSTEM IN THE STATE IT WAS IN BEFORE ANY TAMPERING WAS DONE.

MULTIPLE USE OF 'LOK' WILL NOT RESET THE MEASURED KEYBOARD TIME. THIS CAN ONLY BE DONE AFTER THE CORRESPONDING 'NOK' IS ISSUED. IN OTHER WORDS, AN EMBEDDED SERIES OF 'LOK'S' AND 'NOK'S' (LOK LOK LOK NOK NOK NOK) WILL HAVE THE SAME EFFECT AS THE OUTERMOST PAIR - INNER 'LOK'S' WILL NOT RESET THE KEYBOARD TIME, AND INNER 'NOK'S' WILL NOT TURN THE LOCK STATE OFF SO THAT A SUBSEQUENT 'LOK' COULD RESET THE TIME. NOR CAN A USER INTERRUPT A FUNCTION AND ISSUE HIS OWN 'NOK', THUS ENABLING HIM TO TAMPER WITH THE SYSTEM, REISSUE A 'LOK', AND CONTINUE. IF HE TRIES THIS, THE SYSTEM WILL DETECT THE INTERRUPT AT THE TIME THE 'NOK' IS CALLED. IT SHOULD BE NOTED THAT 'LOK' AND 'NOK' DO NOT DETECT ILLEGAL INTERRUPTS AT THE VERY INSTANT THAT THEY OCCUR. RATHER, THEY ARE DETECTED WHEN FUNCTION EXECUTION RESUMES AND ENCOUNTERS A SUBSEQUENT 'NOK' OR A FILE OPERATION. 'LOK' AND 'NOK', LIKE OTHER SYSTEM FUNCTIONS, DEPEND ON 'SUPERSEAL' TO HIDE INTERNAL VARIABLES FROM THE USER.

GIVEN 'LOK', 'NOK' AND 'SUPERSEAL', A SECURE SYSTEM CAN BE BUILT BY MAKING SURE THAT ALL SNAPSHOTS ARE TAKEN AND PROCESSED ONLY WHILE THE LOCK STATE IS IN EFFECT. THIS MEANS THAT 'LOK' HAS TO BE ISSUED BEFORE A 'READ' OR A 'WRITE', AND 'NOK' HAS TO BE ISSUED AFTER A 'DONE' OR AN 'OUT'. ANY NUMBER OF SNAPSHOTS CAN BE INCLUDED WITHIN ONE 'LOK' - 'NOK' PAIR. WHILE WITHIN THIS LOCK STATE , ALL KEYBOARD INPUT MUST BE ACCEPTED THROUGH THE 'INPUT' FUNCTION.

MAKING A SYSTEM SECURE IS A MODERATE NUISANCE. FOR SYSTEMS WHERE SECURITY IS NOT A PROBLEM IT OUGHT BEST BE IGNORED. SUCH SYSTEMS WOULD BE THOSE WHERE THE STEWARD IS THE ONLY USER OR WHERE ALL USERS CAN BE TRUSTED. IN ALL OTHER CASES, CARE SHOULD BE TAKEN TO BUILD A SECURE SYSTEM.

AN APPLICATION OF THE BINOMIAL DISTRIBUTION TO SALES PLANNING

by

Harley M. Courtney - The University of Texas at Arlington
Carl M. Cheney - Dallas County Community College

ABSTRACT

This paper will present a method of sales budgeting which is particularly applicable to certain companies, will contrast the method with the use of expected value, and will present a set of APL functions which can be used to formulate the sales plan.

Introduction

A variety of companies have been and are experimenting with the use of subjective probablistic forecasts in the budgeting process. The planning literature usually suggests that probabilistic estimates are more useful than are single-value forecasts, and ordinarily suggests an approach utilizing the concept of expected value.(1) While the expected value approach is appropriate for many companies, particularly those having numerous sales such that no single sales transaction is significant, there has been little recognition that the methodology does not yield useful information when there are single transactions constituting a material dollar value. In fact, the use of expected value in these circumstances may seriously mislead management in their formulation of plans for materials acquisition, labor recruitment, and financing, all of which depend upon sales estimates. Companies having relatively few large sales transactions typically manufacture to order; that is, sales effort and the consummation of sales contracts precedes production. These characteristics describe such firms as building contractors, defense contractors, and shipbuilders. The methodology described here is felt to be particularly applicable to such businesses.

An Illustration of the Problem.

Assume that a road builder is planning his resource requirements for the ensuing quarter, and contemplates bidding on the following projects which must be completed during that quarter:

PROJECT NAME	AMOUNT	SUCCESSFUL BID PROB.
Highway 90	$70,000	.4
Flags Amusement Park	30,000	.7
Cato Junction	120,000	.6

If the above tabulation is considered to be a 3 x 2 matrix known as M, the expected value may be defined in APL as

```
+/×/M
```

and is $121,000. Many firms similiar to our illustrative firm, have attempted to acquire resources based upon such sales estimates with disappointing results. The calculation assumes that the number of sales is sufficiently large such that a normal distribution is approximated, and this is often not the case.

Rather, a binomial distribution is present and the calculation presented in Exhibit 1 is appropriate for estimating possible sales levels and resulting resource requirements. It should be noted that there is no possiblity that sales will be $121,000 if only the three contracts are considered. Moreover, the probability of experiencing sales within twenty per cent of the expected value is only .108. And there is a sixteen per cent probability that sales will be about 100,000 greater than the expected value as well as an equal chance that sales will be only $30,000. The tabulation of these possible outcomes permits management to weigh the risk involved in any sales budget adopted, rather than planning for a unique sales level based on the expected value calculation and being unaware of the possible consequences.

Managements presently select contracts on which to bid based on their heuristic feel for the associated risks. Generally, small contractors only bid on large jobs when they have confidence in their ability to acquire resources necessary for performance of any ensuing contractual obligations. To make bids without this capability would risk either suits for non-performance, or overinvestment in resources. The proposed approach reveals the risk associated with certain combinations of contract bids, and permits the selection of "bidable" contract combinations which best satisfy sales requirements and risk aversion. Thus, in the above example, one may make an analysis of the benefits and risks associated with bidding on the first two contracts only, and consequently, decide not to bid on the Cato Junction job.

POSSIBLE SALES LEVELS	PROB. WIN(LOSE) BID HIGHWAY 90	FLAGS PARK	CATO JCN.	PROB. OF OBTAINING SALES LEVEL
0	(.6)	(.3)	(.4)	0.072
30000	(.6)	.7	(.4)	0.168
70000	.4	(.3)	(.4)	0.048
100000	.4	.7	(.4)	0.112
120000	(.6)	(.3)	.6	0.108
150000	(.6)	.7	.6	0.252
190000	.4	(.3)	.6	0.072
220000	.4	.7	.6	0.168
				1.000

Exhibit 1: Calculation of probabilies of various sales levels

Despite the inherent risk, management may elect to bid on all jobs. If so, the binomial analysis will suggest the amount of flexibility which should be incorporated in the plans. For example, the contractor might secure an option to purchase critical materials and do some scouting to locate potential personnell. Such planning flexibility will be costly, and will be avoided when feasible.

Applications of the Concept

The number of calculations required to apply this method of sales forecasting increase exponentially as the number of contracts increases and can become unmanageable with even a small number of contracts. If n is the number of contracts, the number of possible sales levels is

2*N

assuming that all probabilities of obtaining contracts are less than unity. For thirty contracts, there are over one billion combinations. The first few contract numbers and the combinations are shown in Exhibit II. It can be seen that hand calculations are infeasible beyond a few contracts.

A solution proposed by one of the authors (2) is to sort contracts into a few probability catagories, to sum the catagories, and to assume that the sums represent single contracts. One defense contractor has found this method to be superior to expected value calculations for sales planning.

At first glance it seemed that use of the computer would avoid the problem of large numbers of combinations. However, this "pure" approach can be carried only a few contracts further in the normal 32K APL workspace than can be handled by hand. One might have dispaired of a feasible solution except for the discovery that with a relatively few contracts of the same approximate magnitude, present value calculations yield reasonably reliable answers, and that the "larger" contracts in any group under consideration cause the greatest uncertainty. Thus a compromise struck was to calculate the expected value of all but the largest few contracts, and to consider this expected value as constituting a contract with a success probability of one.

Description and Operation of the Model

The model was constructed so as to make it convenient for a planner to use; only two functions are apparent to the user, ENTER and RUN. Exhibit III illustrates the use of ENTER. Contract amounts (usually in thousands) and the subjective probabilities of successful bids are entered and the specific result is a matrix of the data so that the user may check input

NUMBER OF CONTRACTS	*NUMBER OF SALES COMBINATIONS*
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576

Exhibit II: Contracts and sales combinations.

```
      ENTER

PLEASE ENTER EACH CONTRACT AMOUNT FOLLOWED
BY THE CHANCE OF GETTING IT AND A RETURN.
INDICATE YOU ARE FINISHED BY ENTERING 'END'.
⎕:
      5430 .4
⎕:
      380 .25
⎕:
      100 .7
⎕:
      356 .6
⎕:
      89 1.
⎕:
      431 .3
⎕:
      END
 5430          0.4
  380          0.25
  100          0.7
  356          0.6
   89          1
  431          0.3
```

Exhibit III: Illustrative run of data entry function.

accuracy. The matrix result is a global variable DATA. To illustrate the significance and use of the model, three data sets - DATA1, DATA2, AND DATA3 are used in Exhibit IV. Several comments on the algorithm are required. Earlier we observed that the 32K workspace cannot accomodate combinations for more than a few contracts. As a consequence, the following steps were taken:

1. A variable TKN was created and assigned a small value, say 5.
2. All contracts with a probability of one were combined .
3. The remaining contracts were sorted and the expected value of the smaller contracts obtained and added to the certain contracts.
5. A binomial distribution was generated with the TKN largest contracts and the sum of the certain contracts plus the expected value of the smaller ones.

Consider DATA1, for example. The contracts already signed (those with prob. of 1) are summed, the expected value of all remaining contracts excepting the largest 4 (TKN = 4 in this execution) are obtained, and sum of the expected values is added to the obtained contract total. This sum is then considered to be the lowest possible sales level for the ensuing period, and is treated as a single contract. Possible sales levels are then computed using the sales certain and the four largest contracts having a success probability of less than one. TKN may be varied and, with TKN less than or equal to six, the functions will execute in a 32K workspace.

In addition to calcuations of the binomial distribution, cumulative probabilities are obtained. Note that there is a chance of .405 that sales will be reach the expected value of all sales (expected value of the contracts was included in this execution in order to contrast it with the binomial probability). The matrix is presented in graphic form following the numeric data. When TKN is 4, as in this execution, the 16-row table is comprehensible. But when TKN is 5 and the table has 32 rows, the significance of the data is difficult to grasp; therefore, the graphs become essential.

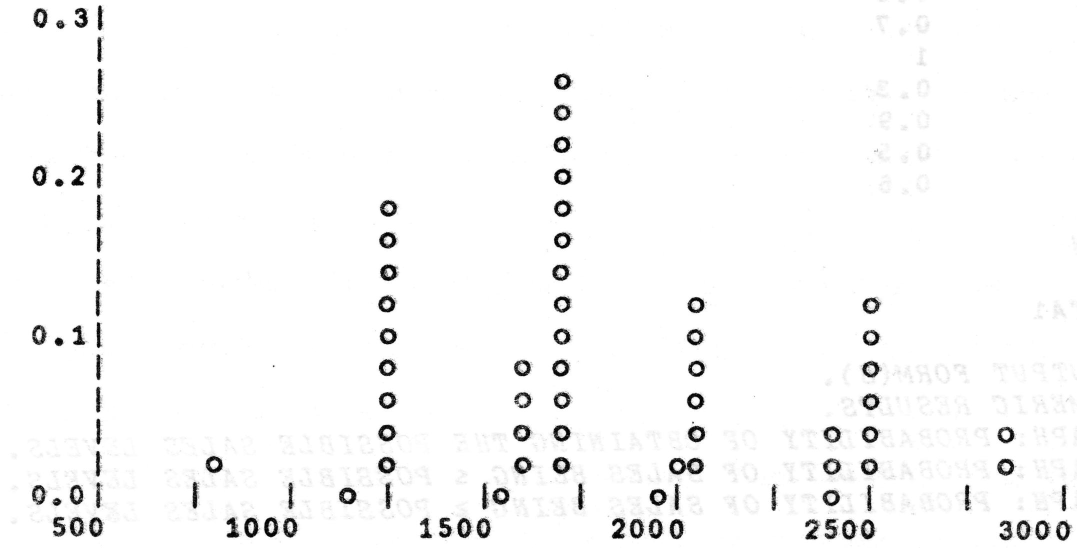
The values in DATA1 were randomly generated. Seldom will potential contract values be so evenly distributed; usually they will evidence increasing degrees of skewness. To illustrate this impact, two randomly selected contracts from DATA1 were changed in amount to 1500 and 2000 and the modified matrix was called DATA2. The execution of RUN using DATA2 illustrates the effect which large potential contracts have on the probability distribution. Expected value of all contracts in DATA2 is 2852 and, as can be seen, this is not in the general area of the most likely total sales figures. Use of expected value for planning sales under these circumstances would very likely be costly either in idle resources or in costs incurred to obtain needed resources.

A management, confronted with the DATA2 situation, would likely search for some other combination of contracts upon which to bid so as to obtain a more tolerable risk situation. DATA3 illustrates one solution: to not bid on the two large contracts. While this may not maximize the expected value of the potential

```
      DATA1
   93             0.9
  163             1
   72             0.7
  366             0.3
  254             0.7
  136             1
  784             0.3
  456             0.9
  350             0.5
  453             0.6

      RUN
⎕:
      DATA1

CHOOSE OUTPUT FORM(S).
   1) NUMERIC RESULTS.
   2) GRAPH: PROBABILITY OF OBTAINING THE POSSIBLE SALES LEVELS.
   3) GRAPH: PROBABILITY OF SALES BEING ≤ POSSIBLE SALES LEVELS.
   4) GRAPH: PROBABILITY OF SALES BEING ≥ POSSIBLE SALES LEVELS.
1234


EXPECTED VALUE OF THE CONTRACTS IS $ 1813


POSSIBLE    PROB.     PROB.      PROB.
  SALES     OBTAIN    SALES      SALES
 LEVELS     LEVELS    ≤LEVEL     ≥LEVEL

   786      0.020     0.020      1.000
  1152      0.008     0.028      0.980
  1239      0.029     0.057      0.972
  1242      0.176     0.234      0.943
  1570      0.008     0.242      0.766
  1605      0.013     0.255      0.758
  1608      0.076     0.330      0.745
  1695      0.265     0.595      0.670
  1936      0.004     0.599      0.405
  2023      0.013     0.611      0.401
  2026      0.076     0.687      0.389
  2061      0.113     0.800      0.313
  2389      0.005     0.806      0.200
  2392      0.032     0.838      0.194
  2479      0.113     0.951      0.162
  2845      0.049     1.000      0.049
```

Exhibit IV: Illustrative use of the model to formulate plans.

PROBABILITY OF OBTAINING THE POSSIBLE SALES LEVELS.



PROBABILITY OF SALES BEING ≤ POSSIBLE SALES LEVELS.

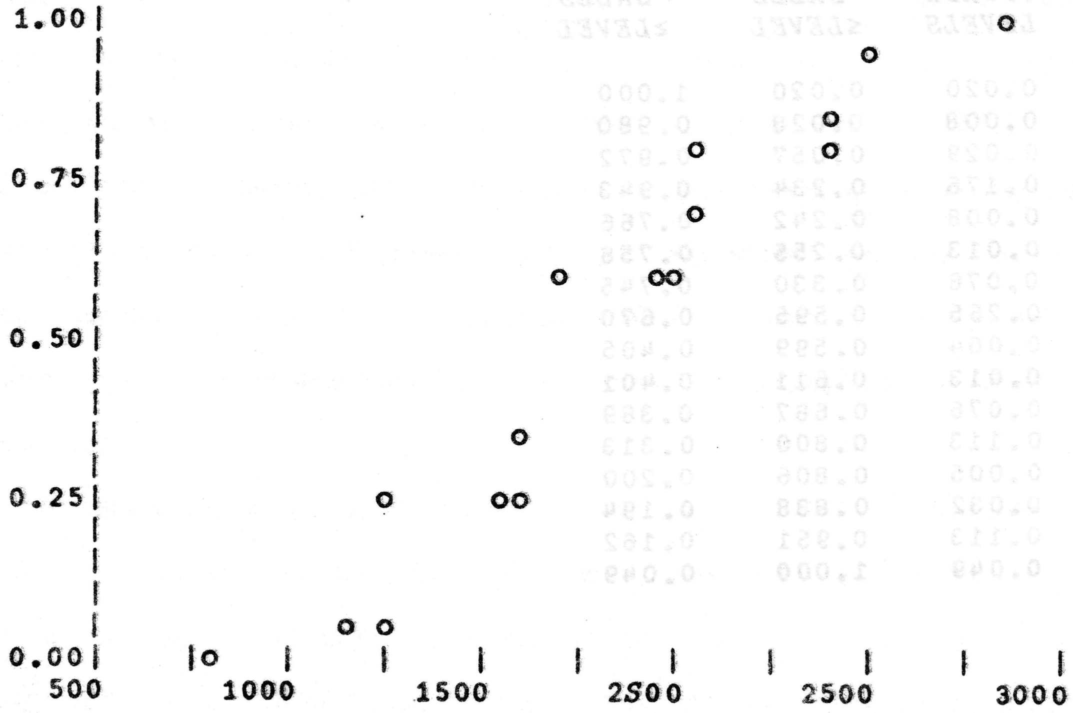


Exhibit IV: Continued.

PROBABILITY OF SALES BEING
≥ POSSIBLE SALES LEVELS.

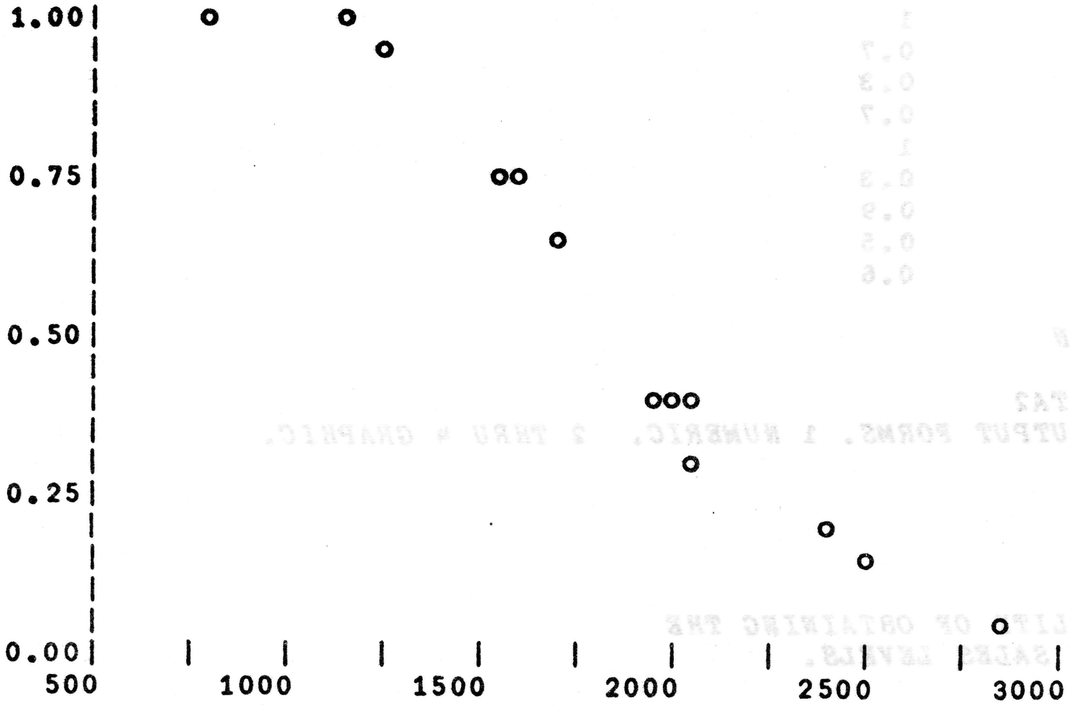


Exhibit IV: Continued

```
DATA2
   93             0.9
  163             1
   72             0.7
  366             0.3
  254             0.7
  136             1
 1500             0.3
  456             0.9
 2000             0.5
  453             0.6

      RUN
⎕:
      DATA2
CHOOSE OUTPUT FORMS. 1 NUMERIC.  2 THRU 4 GRAPHIC.
2



 PROBABILITY OF OBTAINING THE
POSSIBLE SALES LEVELS.
```

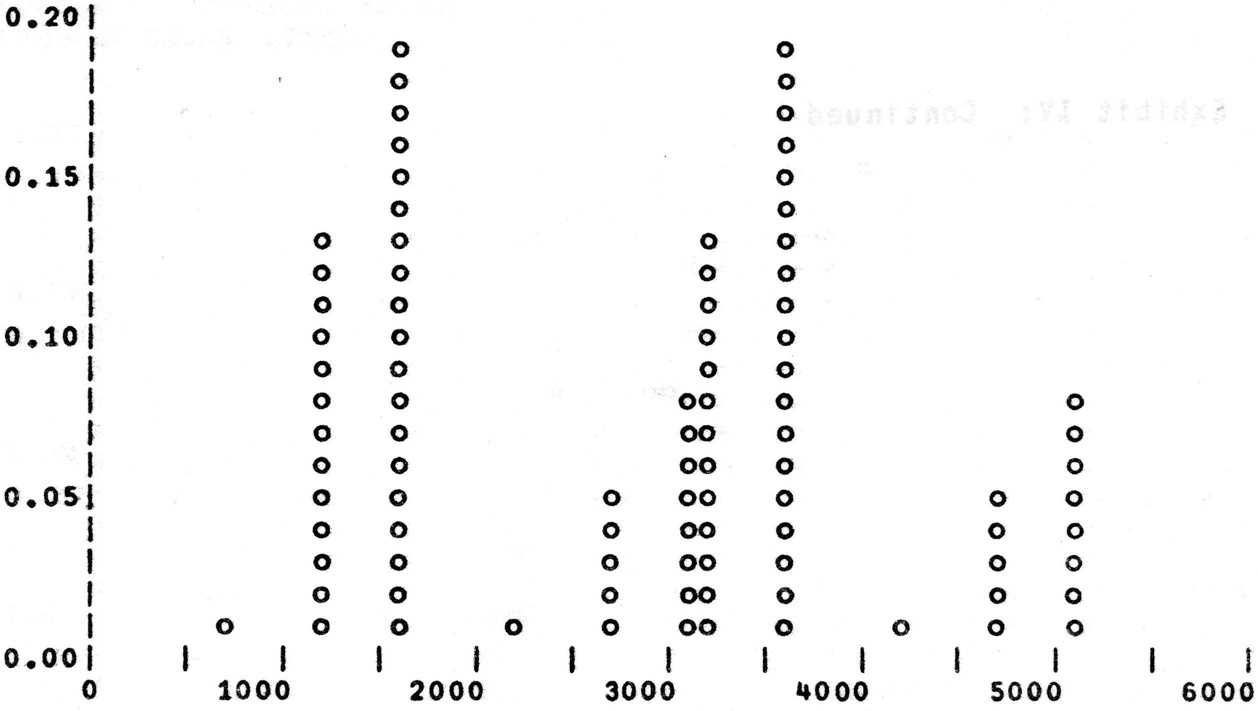


Exhibit IV: Continued.

```
      DATA3
  93            0.9
 163            1
  72            0.7
 366            0.3
 254            0.7
 136            1
   0            0
 456            0.9
   0            0
 453            0.6

      RUN
⎕:
      DATA3
CHOOSE OUTPUT FORMS. 1 NUMERIC.  2 THRU 4 GRAPHIC.
2


 PROBABILITY OF OBTAINING THE
POSSIBLE SALES LEVELS.
```

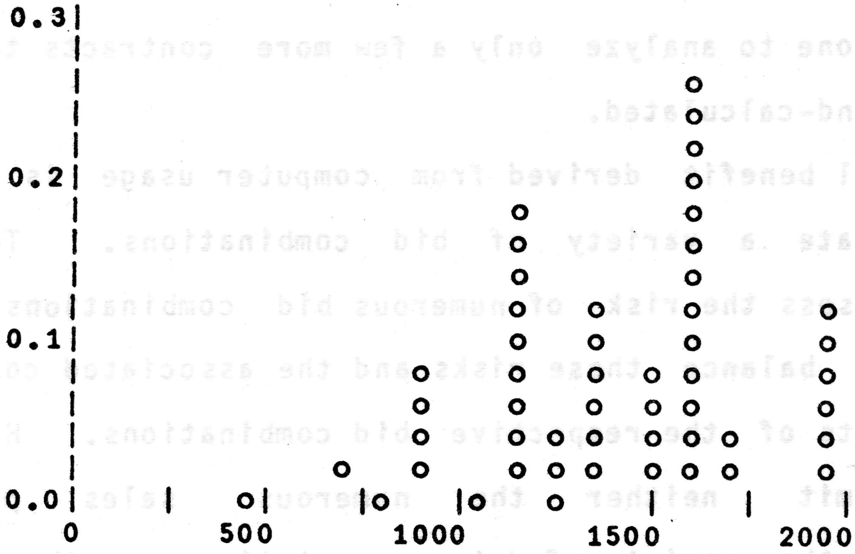


Exhibit IV: Concluded.

contracts, it does increase the probability that the business will survive.

Conclusions

Use of the computer for generating binominal distributions of expected sales was initially expected to permit significantly greater accuracy in the results than could bp obtained by hand calculations. This was not the case for two reasons. As the number of contracts increases, one approaches a normal distribution if they are randomly distributed. If there are few large biddable contracts, only those few represent significant risk, and the smaller ones may be treated using expected value. Secondly, the rate of increase in combinations is such that the computer permits one to analyze only a few more contracts than can be feasibly hand-calculated.

The principal benefit derived from computer usage is the ability to simulate a variety of bid combinations. Thus management can assess the risk of numerous bid combinations on operations and can balance these risks and the associated costs against the benfits of the respective bid combinations. Hand calculations permit neither the numerous sales plan reformulations nor the variety of data presentations as the set of functions illustrated here.

FOOTNOTES

1. Probably the best example is Cost Accounting: A Managerial Emphasis (3rd. ed.) by Charles T. Horngren. Horngren treats expected value for projections throughly, including the calculation of the standard deviation and the variance from expected value. As is typical of the writers, he does not examine the underlying assumption of a normal distribution.

2. Harley M. Courtney and Frederick V. Brooks, "Cumulative Probabilistic Sales Forecasting," Management Accounting, May 1972, pp. 44-47.

APPENDIX

APL FUNCTIONS

```
     ∇ X←ENTER;Y;END
[1]    ''
[2]    END←●○1
[3]    INSTRUCT
[4]    INSTRUCT←'ENTER DATA.  TYPE ''END'' WHEN FINISHED.'
[5]    X← 0 2 ρ0
[6]   LUP:→((END∊Y),2≠ρY←,⎕)/V,LE
[7]    X←X,[1] Y
[8]    →LUP
[9]   LE:'PLEASE ENTER ONLY TWO NUMBERS TO A LINE: THE CONTRACT AMOUNT'
[10]   'FOLLOWED BY THE CHANCE OF GETTING IT.'
[11]   →LUP
[12] V:DATA←X
     ∇
```

```
     ∇ RUN;X;BR;EV
[1]    →ZER×⍳0=1↑ρX←⎕
[2]    EV←⌊+/×/X
[3]    OUTPUT
[4]    OUTPUT←'CHOOSE OUTPUT FORMS. 1 NUMERIC.  2 THRU 4 GRAPHIC.'
[5]    X←EVAL X
[6]    →BR[1]×⍳END≠1↑BR←(('1234'∊⍞)/NUM,GH1,GH2,GH3),END
[7]    'NOW THAT''S SILLY: IF YOU DIDN''T WANT ANY OUTPUT WHY DID YOU'
[8]    'BOTHER TO RUN THIS PROGRAM?  CHOOSE AGAIN...'
[9]    →¯3+⌶26
[10] NUM: 2 1 ρ' '
[11]   'EXPECTED VALUE OF THE CONTRACTS IS $ ';EV
[12]   3 1 ρ' '
[13]   'POSSIBLE    PROB.     PROB.      PROB.'
[14]   '   SALES   OBTAIN     SALES      SALES'
[15]   '  LEVELS   LEVELS    ≤LEVEL     ≥LEVEL'
[16]   ''
[17]   PRT X
[18]   →BR←1↓BR
[19] GH1: 5 1 ρ' ';'PROBABILITY OF OBTAINING THE'
[20]   'POSSIBLE SALES LEVELS.'
[21]   ''
[22]   P2 X[; 1 2]
[23]   →BR←1↓BR
[24] GH2: 5 1 ρ' ';'PROBABILITY OF SALES BEING'
[25]   '≤ POSSIBLE SALES LEVELS.'
[26]   ''
[27]   P34 X[; 1 3]
[28]   →BR←1↓BR
[29] GH3: 5 1 ρ' ';'PROBABILITY OF SALES BEING'
[30]   '≥ POSSIBLE SALES LEVELS.'
[31]   P34 X[; 1 4]
[32] END: 5 1 ρ' '
[33]   →0
[34] ZER:'NOW I MAY JUST BE A DUMB IBM 360,'
[35]   'BUT EVEN I KNOW THAT I NEED SOME DATA'
[36]   'BEFORE I CAN GIVE RESULTS.'
[37]   'START OVER...                 ...SHEESH!'
[38]   ''
[36]   →1
     ∇
```

```
      ∇ X←EVAL Y;Z;A;B
[1]     A←+/(Y[;2]=1)/Y[;1]
[2]     Y←(~Y[;2]∊ 0 1)⌿Y
[3]     A←A++/×/(TKN,0)↓Y←Y[⍒Y[;1];]
[4]     Y←((TKN⌊⍴Y[;1]),2)↑Y
[5]     Z←⍉~~((⍴Y[;1])⍴2)⊤¯1+⍳2*⍴Y[;1]
[6]     X←A++/Z×(⍴Z)⍴Y[;1]
[7]     X←X,[1.5]×/((~Z)×1-Y)+Z×Y←(⍴Z)⍴Y[;2]
[8]     X←X[⍋X[;1];]
[9]     X←X,(SS X[;2]),[1.5]○|SS⌽X[;2]
[10]    A←¯1≠Y←X[;1]=1⌽X[;1]
[11]  LUP:→END×⍳(⍴Y)<Z←Y⍳1
[12]    X[Z; 2 3 4]←+⌿X[B←(Z-1)+⍳((Z-1)↓Y)⍳0; 2 3 4]
[13]    →LUP,A[1↓B]←Y[B]←0
[14]  END:X←A⌿X
      ∇

      ∇ X←SS Y
[1]     X←Y+.×X∘.≤X←⍳⍴Y
[2]     ⍝SUM-SCAN ALGORITHM.  WHEN SUM-SCAN BECOMES AVAILABLE AS
[3]     ⍝A PRIMITIVE FUNCTION, REPLACE LINE ONE WITH:  X←+\Y
      ∇

      ∇ X←PRT Y
[1]     X←'I7,3F9.3' ΔFMT Y
      ∇

      ∇ X←PRT2 Y
[1]     X← 7 0 9 3 9 3 9 3 DFT Y
      ∇

      ∇ P2 X
[1]     H̲S̲←1
[2]     20 60 PLOT X
      ∇

      ∇ P34 X
[1]     H̲S̲←0
[2]     20 60 PLOT X
      ∇
```

Note that HS (histogram) appears in the two immediately preceding functions. It appeared as stmt. 2 in PLOT and was deleted.

```
     ∇ RESET
[1]    INSTRUCT←'
       PLEASE ENTER EACH CONTRACT AMOUNT FOLLOWED
       BY THE CHANCE OF GETTING IT AND A RETURN.
       INDICATE YOU ARE FINISHED BY ENTERING ''END''.'
[2]    OUTPUT←'
       CHOOSE OUTPUT FORM(S).
          1) NUMERIC RESULTS.
          2) GRAPH: PROBABILITY OF OBTAINING THE POSSIBLE SALES LEVELS.
          3) GRAPH: PROBABILITY OF SALES BEING ≤ POSSIBLE SALES LEVELS.
          4) GRAPH: PROBABILITY OF SALES BEING ≥ POSSIBLE SALES LEVELS.
       '
     ∇
```

Administrative Datafile Handling via APL

ABSTRACT: Procedures are described for the computer creation and manipulation of data files containing administrative information such as names, addresses along with coded account status. Via the use of an APL computer program package, variable length files can be entered, updated and sorted according to the user's need. Certain account information can be included in each file which will allow automatic invoicing of overdue accounts. Account status of individual files or of whole data blocks can be accessed rapidly. Other features include, computer printouts of mailing labels, account statements, and formatted file lists on either line printers or remote terminals.

The package is ideally suited for creating and maintaining mailing and subscription lists. Designed for use by secretaries or anyone who can type, the package is easy to use. No prior knowledge of APL is necessary. Only a few system commands need to be learned. Some cost information is included to allow comparison of the computer method and manual method of this type of data handling.

Author: Dr. W.R. Tinga, Dept. of Electrical Engineering, University of Alberta, Edmonton 7, Alberta, Canada.

APL for CAL and Information Management

April 28,1973

David Vaskevitch
Erindale Campus,
University of Toronto.

Introduction:

In this paper I intend to discuss the use of APL for Computer Assisted Learning (CAL) and for the development of Information Management systems. It will be seen that there is a considerable degree of overlap between the two fields.

CAL will be discussed first and a set of functions which has been developed over the last three years will be explained in some detail. Later a new philosophy which has developed will be exposed and the connection of CAL with Information Management shown.

APL/CAT:

A set of functions, called APL/CAT, has been developed at Erindale to facilitate the authoring of conventional lesson material. By "conventional" lesson material is meant the type of material which can be prepared in languages such as COURSEWRITER and other authoring languages. This does not mean that the functions are limited to the capabilities of other languages or that they are modeled after any one particular language, but rather that the author must have at least as much power as he would have with any of the other course authoring languages.

I do not intend to discuss all of the functions in detail herein since it would be outside of the scope of the paper, but a manual on the "language" is available. Functions exist to facilitate input of a student's response and proccessing of this response to see if it matches an expected answer. The functions allow the student to leave the lesson in the middle and return to his point of exit at a later time. In addition housekeeping functions initialise all variables and collect the student's full name separating the first name from the rest for subsequent use by the lesson. In order to illustrate the flexibility and ease of use of the package the answer proccessing and branching statement will be discussed in detail.

Answer proccessing assumes that the students response is contained in a standard global variable. If the author has used the provided input functions then he need not worry about this detail. However if he wishes he can use his own input mechanism or manipulate the input response by using this variable. The comparison of the student's response with the anticipated answer is handled by five routines which may be combined on a single line in a manner resulting in a statement closely paralleling the natural language in which the author might state the comparison. The result of a comparison is boolean.

The five functions which may be combined to state the comparison are: SCAN, MATCH, AND, OR, BEFORE. SCAN and MATCH are monadic while AND, OR, and BEFORE are dyadic. Clearly the syntax of APL demands that in any given comparison expression the leftmost function invoked be either SCAN or MATCH while the functions to the right may be any combination of AND's, OR's, and BEFORE's. The actual arguments given by the author are literal. Several examples of comparison expressions are shown below:

```
SCAN 'GEORGE' BEFORE 'WASHINGTON'

MATCH 'QUICK' OR 'FAST' OR 'RAPID'

SCAN  'COLUMBUS' OR 'ERICSON' BEFORE 'DISCOVERED' OR 'FOUND'
AND 'NORTH' BEFORE 'AMERICA'
```

The result of the three examples should be fairly obvious. The MATCH function looks for an exact match. Extraneous characters, such as blanks, will cause the response to be a mismatch. If it is desired to use match editing functions are included in the package which will cause the input functions to automatically remove designated characters, or replace designated characters with other designated characters.

SCAN looks for any occurrence of the anticipated answer in the response. In fact if the anticipated answer is considered as a given string and the response as the target string then it scans for any occurrence of the given string in the target string.

The last example brings up the question of the "order" of execution of the comparison. While APL executes from right to left APL/CAT, by devious means, gives the effect of not doing so. OR is evaluated first, followed by BEFORE, followed by AND. An explanation of the reasoning behind the choice of the order of execution is in order. It is assumed that the author is looking for a given sentence. First he is looking for a number of phrases which may occur in any order as long as they are all present. Hence AND is evaluated last. Each phrase consists of a number of words which must occur in a given order. This is checked by BEFORE. Finally each of these words has a number of "synonyms" by which it may be replaced without affecting the correctness of the phrase. This check is performed by OR.

It is possible to alter the given order of execution by using bracketing. However it must be kept in mind that each expression must be terminated at the left by SCAN or MATCH. The other three routines, alone, may not be bracketed. An example of correct bracketing is shown:

```
(SCAN 'PINK' BEFORE 'PANTHER')v SCAN 'THIEF' OR 'THEIF'
```

An interesting additions to the set of comparison function is given by the NULL function which is used to designate a null character. The presence of a null character in an argument of a comparison function indicates that any character may be in that position of the student's response. Hence if we had:

```
NULL '*'

SCAN 'TH**F'
```

the computer would accept any five letter word having "TH" as the first two letters and "F" as the last letter.

As mentioned above SCAN and MATCH return boolean results. The branching function accepts a boolean result. The syntax is:

```
X IF Y
```

where X is a vector of statement numbers or labels of lines in the lesson and Y is a boolean value. The function returns an explicit result which is the number of a line in the lesson. It is intended to be used to the right of the branch arrow:

```
-X IF Y
```

If Y is 0 then the lesson continues on the next line. Otherwise the first time that the line is encountered the computer branches to the first line number in X; the second time to the second line number; and so on until it encounters the last element in X in which case it continues branching to this element indefinitely.

A sample combination comparison and branching statement is shown below:

```
-(HINT1,HINT2,HINT3,WRONG) IF SCAN 'THEIF'
```

If THEIF is not found in the response then the lesson continues to the next line of the lesson. Otherwise the first time that THEIF is found the lesson continues with the line labellexo"HINT1"; the second time THEIF is found the lesson continues with HINT2; the third time with HINT3; and the fourth and all other times with the line labelled WRONG.

The above gives an idea of the flexibility and ease of use of APL/CAT. It is by no means a complete exposition of answer proccessing in APL/CAT and does not touch on any of the other features of the package.

It is interesting to compare APL/CAT with other authoring languages. We have found it to be as powerful and easy to use as any other language in current use. Furthermore it is clear that if additional features were desired they could be added on by writing additional functions or changing the old ones. None of the functions is more than ten lines in length indicating that they could be easily changed by someone familiar with APL.

Because of the workspace orientation of APL each workspace in our library contains not only the lesson but also the functions used in the lesson. Hence when we change APL/CAT this does not affect previously written lessons. We can say that all of the lessons we have ever written still work.

APL/CAT has been found to be very easy to learn. Authors with virtually no experience with APL or computers have found APL/CAT easy to use and were able to write lessons very quickly without having to become highly familiar with APL first.

Clearly APL/CAT demostrates that APL is viable as a course authoring language. It shows that it is possible to do as much as easily in APL as in other course authoring languages. Furthermore because of the fact that APL/CAT is written in APL each author is able to develop his own special version of APL/CAT suited to his own needs. This is certainly not possible in most authoring languages.

While APL/CAT facilitates the authoring of conventional material another type of "lesson" is being developed at Erindale. The conventional type of lesson material might be labelled, in a manner of speaking, as "page turning". The lesson produced by an author is, in many cases, very similar to a programmed text book. The student is passive, only having to supply answers from time to time in order to show that he is paying attention. Furthermore, except for answer comparison, conventional lesson material uses very little of the computer's power except as a storage medium. We have been trying to develop a type of lesson which would differ radically from the more conventional page turner. It is easiest to illustrate the basic difference with an example.

The author has just completed a THEOREM PROVER for the Propositional Logic. The package allows the student to propose a theorem which he will try to prove. The student then enters his proof a line at a time. He is allowed to continue as long as what he does is valid. When he makes a mistake the computer will immediately point out his mistake and erase the line. At any time the student may ask the computer to prove any proposition for him. Hence he could have the computer prove the entire theorem. Alternately he could ask for as much, or as little, help as he wants and needs. The proofs produced by the computer are constructive (intuitionistically valid) when this is possible. Such a package is much closer to a tutor than to a page turner. The student is not passive but is actually directing the computer instead of being directed by the computer.

It goes without saying that such a package as described above represents a significant use of the computer's problem solving capability. Interestingly in such an application the computer is hardly used at all as a storage medium. This provides a conclusive answer to one of the questions of critics of CAL: Could not the computer be replaced by a sophisticated slide projector, or other form of "teachine maching"?. In the case of the theorem prover the answer is irrefutably "no". Further the theorem prover is not even a teacher. Rather it is a tutor helping the student with his problems but not doing the thinking for him.

Several packages of this nature have been developed at Erindale. One is an electric circuit package. The student proposed a two dimensional network and procceeds to solve it. He is not allowed to make errors. The solution is algebraic, as opposed to numeric. At any time he may have the computer solve any subcircuit for him, up to and including the entire network. Thus, again, he can get as much or as little help as he likes.

Another package of this nature is a rational matrix arithmetic workspace. Here a student sets up, with the help of a function, a matrix containing rational entries. Then he can have the computer perform row operations for him and hence manipulate the matrix as he sees fit. This package is very useful in teaching algebra or linear programming and is one of the most heavily used packages we have.

It will be seen that the above applications are of such a nature that a language with the power of APL is required. Many authoring langcages would be inadequate for developing any one of these packages. At the same time I am convinced that if CAL is to achelve the importance that it can and fulfill its useful role in teaching and learning then it is packages such as those described above which will become representative of CAL. The conclusion is that not only can APL do as much as other authoring languages but it can do things, which must be done, which many authoring languages cannot do.

Information Management:

In this section I intend to discuss several packages, currently under active development, which fall under the category of Information Management systems. Later in the paper I will discuss Information Management in relation to CAL.

Information Management Systems fall into two broad classes, with a considerable amount of overlap. One type assumes a certain type of organisation in the database. Further it is assumed that the data is preordered into a format conformable with the type of data structure which the system is designed to handle. Typically the data must be divided into certain types of fields which, within set bounds, may vary The disadvantage of such a system is that it imposes its own bounds on the data which it is handling. Later if the information required from the data base changes from the initial requirements it may be very difficult for the system to adapt itself.

The second class of system attempts to handle data which is in as natural a form as possible. Typically the initial input might be in the form of natural language text. The system then attempts to abstract the required information from the input. The limitations of this type of system lie in the fact that it is currently impossible for a computer to interpret very adequately natural languages.

The approach taken by the author is to tend to veer towards the natural language approach. Two systems will be described, one of which is a smaller version of the other.

The first system is oriented towards bibliographical applications. The packages resides in two workspaces. One workspace contains a complete document preparation package. This includes an editor and formatter. The editor allows the author to enter and correct text until he has a satisfactory final version. The formatter can then right adjust the text for visual aestheticity. The size of any given peice of text =s limited by the workspace to approximately ten thousand characters. In most cases this means approximately three to four pages of text. The text may be entered as at a normal typewriter and the formatter will still operate successfully. Special control characters may be imbedded in the text to tell the formatter to begin and end formatting in order to preserve tables and other items in the text. The user also has access to files in the system. There are a certain number of editing commands which correspond to "system commands" allowing the user to tie and untie files. Each file must have a format conforming to the requirements of the package. The user may create a file using another command and this ensures such conformity. Other commands allow documents to be saved, deleted, retreived, and replaced by current text. All documents are referenced by name, which is assigned unambiguously by the author at the time of saving the document. This part of the package in itself is a valuable aid to documentation. It allows the user to create and format documentation of a workspace. Then he can put functions in the wo'kspace!being documented to retreive the documentation.

Another workspace contains the function required to create and update indeces. Each index may be constructed "over" a list of d/cuments. The list may contain names of documents from one or more files. An index contains references to every non-trivial word in every document in the list over which it is created. Triviality of a word may be determined by a frequency method or by omitting all words in a given list of words or by both techniques. What is being constructed is, in effect, the heart of a KWIC index. Each index resides in a separate file. Other functions in the workspace allow an index to be changed, deleted, and updated.

A third workspace contains the retreival functions. These functions allow an individual document to be located through the indices and then retreived. The queries which may be directed to the system with respect to any given index allow the user to locate any document with occurences of any group of phrases occuring with a specified frequency. The queries are constructed in a manner closely approximating natural language queries.

The second system is a much larger system which takes off where the last system stops. It is called AIMS for APL Information Management System. There are several intended goals. One of the most important is to make the machine as transparent as possible to the user. Hence there are no limitations on document size as this would make the user aware of a limitation of the machine.

In order ot use AIMS a user must register with the system. Upon registration a file is automatically created for him. For security purposes the user is locked out of his own file. Thus his only access to his file, and the other files in the system, is through AIMS. Naturally there is a system steward who can access all files in case of difficulty.

The basic unit in AIMS is the document. A user may create, edit, and format a document using one workspace of the system. The document may then be saved in the user's private AIMS library. Initially only the user himself has access to the document. He may give other users access in one of two ways: giving access and sending the document as a message.

At any given moment the document which the user is actually working on is called his active, or current, document. The active document may be sent as a message to any other users of AIMS. Upon transmission a copy is created giving the time and date of transmission, the identity of the sender, and the list of recipients. This copy is held until all adressees have received the document. Upon receipt of a document the document replaces the current document and may be handled as any other document may be handled. Any message may be registered with respect to any given adressee(s). If registered the sender is notified when the adressee receives the message and told the time and date of receipt. The workspace containing the transmission and receipt facilities also contains a number of functions which allow the user to determine the status of his receive and transmit queue at any given moment. He can then select to receive certain messages first if he wishes.

The system has been found to lend itself very well to conferences. Furthermore it would form an excellent base on which to build multi-terminal simulations and games.

Each user can give access to his document to any other user(s). He may specify the precise degree of access of any given user and may impose a password which the user must use to access the document.

There are two mechanisms for accessing any document. The document may be copied. This creates a snapshot of the document at the moment of copying. Once copied a document may not be put back into the library under the same name. The user may delete the document and then save the copy under the same name if he wishes and has the permission to do so.

A document may also be captured. This also makes a copy of the document. It also prevents any other user from capturing or deleting the document as long as it remains captured. A captured document may replace its namesake. This prevents file race conditions. A capture is automatically dissolved when another document is retrieved by the user. Other users may still copy the library copy of the captured document even though they may not capture or delete it. Naturally any changes made to the captured copy of the document do not affect the library copy until a replacement is made.

In addition to documents users may create lists. A list is a list of document and list names. The names may be the names of any lists and documents of any users to which the user has access. Lists may also be saved and shared with other users although they may not be transmitted as messages. By having lists refer to lists very complex tree structures may be built up.

Finally indeces may be creajed over lists. Each index consists of two specifications: the name of a list; and a specification of how the fields of the index are to be picked out of the documents. If wehad documents describing each of the students in a school we might specify this as a list. Then if we wanted to we could have a specification telling the computer to pick the english mark field of all polish students with marks of over seventy in mathematics. The index would actually be an index to marks in this case. Currently the index must be specified in a notation developed for the purpose but an unquote operator will simplify things tremendously. When copying or capturing a list one is copying or capturing, respectively, all of the documents contained in the list. This also applies to indeces since these involve lists.

Finally there is a set of functions for accessing and using lists and indeces which allow queries approximating natural language queries.

Several of the cheif features of AIMS are its ease of use, it comprehensiveness, and the high security it affords. Except for failure of APL or the computer it always preserves data integrity and attempts to prevent tampering.

INFORMATION MANAGEMENT AND CAL:

The presence of an adequate information management is crucial to the development of CAL. I will try to point out several important applications below.

It is important to be able to save student responses for later analysis. APL/CAT allows responses to be saved automatically under the following three categories: never; all responses; all unanticipated responses. This can be done by simply copying a different version of the APL/CAT functions into the workspace containing the lesson. A switch is then set by the author determining the mode in whic the saving is to operate. Each response saved can have any number of preselected flags associated with it for later sorting.

An information management system will allow classification of responses according to the content of the responses.

A proctor is under active development. Such a function would help the student find just what lesson he wants. It would eliminate the need for a student to ever type system commands; normally a weak point of APL for CAL. Using the first information management system described each author prepares descriptions of his lesson. Indices are then created and updated. The student can then, through the proctor, find the material pertaining to the subject of his interest. With the indices he finds the relevant material and with the descriptions he verifies his choice.

The proctor can then get the lesson he has decided upon for him. If the student leaves the lesson in the middle the proctor will remember where he left off and return him, if he wishes, at a later time. 6here an unquote operator would be invaluable to maintain the state of variables in the lesson workspace. At another level the indices might reference all material available, instead of just lesson material. If the lesson material contains large quantities of text then an editor could profitably be used to prepare the text. In such a case the text would be accessible to functions. In any case with an unquote the contents of functions would be accessible to other functions. Then the indeces could be prepared based on lesson content as well as the description of the lesson prepared by the author.

The concept of a proctor opens up a new field: COMPUTER MANAGED INSTRUCTION (CMI). With CMI the students learn by themselves and the computer is used to administer and guide them. This is particularly applicable to modular courses. The computer can be used to administer tests and generally keep track of the students in a course. At critical points it can tell the student where to continue. Instead of administering conventional fixed tests the computer can give tailored tests in which the test adapts itself to the student according to his performance. Such tests have the potential of being far more accurate and useful then conventional tests. The computer plays the dual role, in CMI, of guiding the students and keeping the teachers informed as to the progress of the students. Clearly an information management system could play a crucial role here.

Ideally in a modular course the computer could guide the students through the modules and point them to the appropriate resources to help them learn the material. The teachers would be one resource and CAL might be another out of the total resource library.

While the above has not even touched the field of applications where CAL and information management overlap the examples have illustrated that the interaction between the two is fruitful. It is my firm beleif that this is one more aspect in which APL distinguishes itself as language in comparison to other authoring languages. The fact that it is possible to implement such a plethora of information management systems is on more argument in favor of using APL as an authoring language.

CONCLUSION:

We have seen above that APL is a very useful language for both CAL and Information Management. In exploring the interaction between these two we found it to be of great importance to the author of CAL material to have access to an Information Management system and that such access opened up many new possibilities.

I have not discussed the cost of using APL for CAL because, while it is not high, it is rapidly diminishing. With the advent of computers which are morely closely oriented to running APL costs should go down considerably.

At any rate APL is very useful as a language for developing both CAL and Information Management. Its usefulness will increase even more with the implementation of several features such as an unquote operator. However I would strongly advocate continued experimentation with the use of APL for both CAL and Information Management.

APL/11 - A POOR MAN'S APL

D.G. Moffat
M.A. Jenkins
Queen's University

1. INTRODUCTION

A major difficulty with APL is that it is too expensive. To help solve this problem we are implementing an APL system to run on the PDP-11 family of mini-computers, specifically the PDP-11/50 and 11/45. We expect the system to be able to handle over 30 users on-line simultaneously with good response time at a cost per user of about $1/hr. for connect time and 1-2¢/sec. for CPU time. We expect the system to be easily tailorable from a very small, inexpensive single user system to a very large system capable of handling 80 ports or more and having on-line storage for over 4000 workspaces.

2. THE PDP-11 AS AN APL MACHINE

In many ways the PDP-11 is ideally suited to running APL. It is a powerful 16 bit mini-computer with 6 general registers and 2 special purpose registers (the program counter and a hardware stack pointer). It is not overburdened with instructions which add little to the speed of APL but which add a lot to the cost of the machine.

The PDP-11 is a byte addressing machine, unlike most mini-computers which are word addressed. A program can directly access a full 64k byte address space. There is no need for tables of address pointers as in some mini-computers.

The addressing modes are designed to take full advantage of the 8 registers on the machine. In particular there is a mode (auto-increment) which accesses the data whose address is in a register and then increments the register to point at the next piece of data. A second mode (autodecrement) decrements the register first and then accesses the memory location addressed. The amount of the increment, or decrement, depends on the type of data accessed, i.e. 1 for a byte, 2 for a word, 4 for a short real number, 8 for a long real number. Using these addressing modes the APL interpreter can scan down vectors of data without needing code to specifically step the pointers. In particular, autoincrement coupled with byte addressing result in small, fast routines for character processing.

Furthermore, since autodecrement decrements the register before accessing the data while autoincrement increments the register after, these two modes can be combined for stack manipulation as there is no need for special code to keep track of the top of the stack. In the APL implementation we use the hardware stack to store subroutine return addresses and one of the general purpose registers to point to the top of the workspace stack.

The PDP-11 has, as an option, memory relocation hardware. This hardware will separately relocate instruction fetches and data fetches.

Although a given program can only address 2^{16} bytes of instruction and 2^{16} bytes of data, the relocation hardware, and the disks, can address 2^{18} bytes of core. This means that the interpreter can be working on one workspace in core while the swapping disk is writing another workspace into a different part of core. To switch control to the new workspace the data relocation registers are changed to point at the appropriate area of core. Therefore, there is no necessity to maintain a workspace base register. The interpreter sees a single workspace at a fixed logical address allowing straightforward coding techniques using absolute addresses. Because of the relocation hardware, the workspace can be swapped out and brought back in at a different physical location in core.

The I/O structure of the PDP-11 is very convenient. Instead of I/O instructions, the various device control registers and character buffers are assigned addresses and are read from and written into as if they were core memory locations. In fact, memory modules are merely a different type of I/O device. Each terminal port is assigned it's own set of addresses and can cause the CPU to trap to a unique address when an operation is finished. This removes the need for polling of lines.

The CPU is not the only device which can initiate these 'I/O' operations. The disk controller issues direct read/write commands to the core modules thus doing mass transfers without interfering with the CPU.

If the line handling overhead becomes too high it is a simple matter to add a front end processor (a smaller model of the PDP-11) which collects a whole line of input, does visual fiedlity checking on it and deposits it into core without disturbing the main CPU.

The PDP-11/45 also has, as an option, a floating point processor. This unit will do ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPARE, LOAD, STORE, etc. on 32 bit or 64 bit floating point numbers making use of 6 floating point accumulators. It has instructions for floating integers and fixing reals and for separating the fractional part of floating point numbers (very useful for doing fuzz). It will either round or truncate any discarded bits, under program control, and has the usual complement of arithmetic traps. The execution speed of the unit is about 3 times as fast as a 370/145 doing equivalent operations and this is the main reason why APL runs so well on the 11/45.

3. DESIGN OF APL/11

APL/11 is designed as a two layer system: the interpreter and the monitor. The interpreter assumes that there is only one workspace in existence and that this workspace is fixed in core at data space address 8192 and up. The bottom 8k bytes of core (in the data space) contains interpreter constants such as branch tables, identity values, error messages, etc. The interpreter works away on the workspace until it needs some input or has a line of output. At this point it traps to the monitor. When control returns the desired line of input has appeared in the workspace buffer or disappeared from it. The interpreter then continues with its processing.

On top of the interpreter is the APL monitor. This program keeps track of where each workspace is (either in core or on the disk), handles the teleprocessing lines, does such functions as)LOAD,)SAVE,)COPY etc. and handles the clock and accounting. When the interpreter traps to the monitor for some I/O operation the monitor queues up the request and swaps out the workspace. This involves saving 7 registers, saving the return address (to the interpreter) and writing the workspace out onto the swapping disk. When the I/O operation is complete the monitor swaps in the workspace, sets the data space relocation registers to address the workspace, reloads the registers and passes control back to the interpreter at the saved return address. If a read was requested the monitor will also copy the input line from its buffers into the workspace buffer.

If a workspace runs too long without doing I/O then the monitor will be interrupted by the clock. The workspace will be swapped out as above and entered in the list of workspaces which have finished output and are waiting for control.

The interpreter uses a top-down parsing technique to analyze APL lines. A line of input is first passed through a lexical analyser which converts names to symbol table locations, numbers to internal floating point and anything else to a contiguous set of internal codes. The interpreter executes this line of code by scanning it from right to left using the internal codes as offsets into branch tables. Bracketing is handled by having the interpreter call itself recursively and return with a value and a flag specifying the cause of the return. Syntax errors, such as unmatching brackets, are detected by checking for a valid cause of return, e.g. if the interpreter recursed because it saw a) then the only valid cause of return is seeing a (. Any other cause, such as [,; or end-of-line, is an error.

The system is being written in PDP-11 assembler. As an initial implementation aid and to save space in the very smallest version of the system, we are implementing some of the more complicated operators as calls to resident APL functions. These functions perform the necessary operations using that subset of operators which is implemented in assembler. To assist these functions we supply a small set of new operators to do things which are logically straightforward but slow if implemented in regular APL. For example, we have operators to do ×/ρ*X* (i.e. get the number of items in *X*). TYPE *X* (0=> numeric, 1=> character), ρρ*X* (dimensionality of *X*), (ρ*X*) [*I*] (size of *I*'th dimension of *X*) etc. These are operators needed to manipulate data entities as opposed to operators which manipulate the data values.

4. PERFORMANCE

The implementation is far enough along that we can write small functions and compare their execution speed against the speed of other APL systems. These test functions when run on several APL systems, give us an idea of the speed of the operations which are already implemented. It would appear that APL on the PDP-11/45, with a floating point unit, is slightly faster than APL on the B6700 and significantly faster than APL on the 370/145. For example, we wrote a function to find a root of x^2-2 using Newton's method. It was called 1000 times by a second function and timed using my watch. The same problem was then run on the B6700 at Queen's and on the 370/145 at I.P. SHARP in Toronto and timed using I21. The 11/45 ran about 10% faster than the B6700 and 30% faster than the 370/145. As another example, the following table (fig. 1) is composite of measurements made by myself and measurements made by Pierre J. du Bois, Jr., and Jeanne T. Martin of Lawrence Livermore Laboratory, Livermore, California. The table shows the execution times of ACKERMANN's function, with various parameters, when run on several APL systems. The 360/50 and 360/91 are IBM's standard release, the CDC7600 APL is a system written at Livermore in the LRLTRAN language, the 370/145 APL is from I.P. SHART in Toronto, the B6700 APL is from Burroughs Corp. and is written in B6700 ALGOL. The 11/45 APL is our implementation.

FUNCTION	360/50	B6700	370/145	11/45	360/91*	CDC7600
2 ACK 0	167	47	48	35	16.7	10.3
2 ACK 1	350	140	125	90	33.0	24.25
2 ACK 2	750	260	242	170	83.3	65.48
3 ACK 0	467	150	140	95	50.0	30.6
4 ACK 0	3200	1020	940	660	333.3	247.14

*The timing data for the 360/91 is suspect as it looks suspiciously like one or two clock ticks.

Figure 1 - Execution Time in Milliseconds.

The figures for the 360/50, 360/91 and CDC7600 are from SIGPLAN Notices, Vol. 6, Number 9, October 1971, p. 96. The figures for the 370/145, B6700 and PDP-11/45 are our own measurements.

5. COST EVALUATION

The great advantage of APL/11 is the low hardware cost. The basic PDP-11/45 with 16k words of core costs approximately $26,000. The floating point unit costs $4,900; the memory segmentation unit costs $3,900; core costs about $4,500 for 8k words. We estimate that the smallest single user system (PDP-11/40 processor, 8-10k byte workspace, software floating point, model 38 teletype, cassette tape for workspace storate) should cost under $25,000. This is high when compared with single user BASIC systems but not bad when compared with the yearly cost of commerically available APL.

A medium sized system (16 ports, 48k byte workspaces, on-line storage for 200-300 workspaces, one swap area) should cost in the region of $70,000 or about $4,500/port.

The largest system we can envisage (80 ports, 48k byte workspaces, on-line storage for around 4000 workspaces, 3 swap slots) should cost around $200,000 or about $2,500/port.

6. CONCLUSIONS

We believe that a multi-user APL system running on the PDP-11/45 is a perfectly feasible project. The reason that APL currently is so expensive is that the current systems run on very large computers offering too much of one facility and not enough of another. (i.e. the ability to address 16 MILLION bytes of core is a useless facility if all you need is 256k bytes). We believe that the PDP-11/45 offers a much more reasonable balance of facilities at a very reasonable cost.

Our current objective is to produce a cheap, in-house system for our own use. However, if enough outside interest is shown we will consider making the system publicly available.

Simulating Computers in Symbolic Assembler Notation

Dr. Harry J. Saal
Visiting Associate Professor
Computer Science Department
SUNY at Buffalo

Previously published examples of the construction of APL interpreters for computers process a bit representation of the simulated instructions, produced via an assembler or by hand. If one is interested in simulating the effect of each instruction and not the actual fetch and decode, one can simulate a "Symbolic program" instead. This symbolic form is an APL function which is almost identical to the conventional notation used by assemblers. We assume that users of simulators constructed in this fashion are familiar with APL; consequently they can utilize (directly) the debugging, tracing, input/output and computational facilities of APL while programming for the simulated computer. The limitations of this approach and its benefits will be demonstrated using as an example a simulator written for the INTEL MCS-8 Microcomputer.

Simulating Computers in Symbolic Assembler Notation

This note concerns itself with a programming technique of particular interest to APL Users. I am certain that other APL users have independently utilized the method to be described, yet it does not appear to be "common knowledge" to the entire community.

Designers, users and instructors of digital computer systems may find it useful to software simulate (in the broadest sense) some processor.

Ones motivation may derive from a need to experiment with a system prior to its implementation, or from providing facilities in a simulator which are not present in any physical realization. For example, the system to be simulated may have read-only memories which we do not wish to produce until their contents have been substantially debugged. Additionally, we may provide monitoring and measurement capabilities in the simulation that the actual system does not possess. In some cases the system to be simulated will never be constructed, but is used as a teaching tool.

I will not attempt to list the myriad of factors which must be examined in order to choose among the special-and general-purpose languages that are available to anyone faced with implementing a simulator. Instead, I will describe some techniques available in APL, and the limitations which should be kept in mind when evaluating APL as one of the alternatives.

The examples in this paper are taken from a simulator for the INTEL MCS-8 microprocessor chip [1]. The simulator had to provide a convenient means of preparing programs and debugging them in a symbolic form. Figure 1 is a sample program written in symbolic form.

The MCS-8 program is actually executable APL! No assembly (encoding to bits) and interpretation (decoding) is explicitly performed. The syntax of APL is capable of executing statements resembling a symbolic assembler, where the operation codes are APL function names.

We immediately derive the ability to use symbolic labels, editing of programs in symbolic form, tracing and stopping on individual statements from the APL system itself.

One cannot go too far in the comparison with assembler code before explicit knowledge of the APL mechanisms at hand are required. For instances, a conventional assembler statement

```
        LD  REG+1,WHO
```

corresponds to the APL statement

```
        LD  (REG+1),WHO
```

due to the lack of operator precedence in APL. APL will "allow" a user to choose the name of an opcode (APL function) as a label, and proceed to push down the function name without any special indication. Suddenly the user is faced with SYNTAX ERROR messages from lines of code which appear valid to anyone unaware of this trap in APL. Thus a naive user can misuse this approach if he takes too seriously the idea that he is writing "standard" assembler code.

APL's interpretation of "visual fidelity" unfortunately does not include the reproduction of the text-body of a function as it was entered. There is no means of listing a function (such as Figure 1) in a neatly formatted way unless one has the ability to quote functions to characters and unquote characters to functions. Equally striking is APL/360's "cleverness" in accepting

```
[1]  LR  3,4
```

but displaying it as

```
[1]  LR  3 4
```

APL's inability to accept comment information on the same line with an expression is another annoying restriction to users of the scheme described here.

One cannot simulate self-modifying code, or the action of a compiler straightforwardly since no bit patterns are interpreted. This restriction (benefit) has not been relevant in any of the systems we have simulated.

Simulating an interrupt system poses some problems. The executing APL function will proceed from line to line unless an APL transfer of control (signalled by →) occurs. Thus all JMP operations, subroutine calls, etc. must have the right arrow as part of the operation code. If one desired the ability to interrupt the flow of control on any instruction , then all operation codes would need an explicit right arrow. The scheme we have implemented simply delays an interrupt until some jump or call type instruction is fetching when it can then take effect.

Since the code being simulated is actually an APL function, there is no restriction against inserting APL code within the body of the program. This have been very useful for console input/output, calling snap-dumps, or performing complicated algorithms in APL quickly rather than simulating the identical function. We can thus provide a simplified multi-level simulation by interspersing these two languages as desired.

In summary, we hope that users of APL who have to provide machine simulators consider this approach. It results in a package of great convenience due to its symbolic nature and the accessibility of APL debugging tools. Current APL implementations suffer from the inability of the writer of a package to control the environment, error handling etc. as he chooses. But given a modicum of an introduction to APL dealing with a simulator constructed as described seems straightforward to non-APL buffs.

References

[1] 8008 8-Bit Parallel Central Processor Unit; INTEL Corp., Santa Clara, California (June 1972).

```
     ∇ MCS8
[1]    LD L,IMM 2
[2]   LOOP:LD A,IMM 0
[3]    CP M
[4]    →CAL Z,PRINT
[5]    IN L
[6]    →J NZ,LOOP
[7]    →HLT
[8]   PRINT:LD A,L
[9]    OUT 8
[10]   LD B,L
[11]  SET:LD M,IMM 1
[12]   AD B
[13]   LD L,A
[14]   →J NC,SET
[15]   LD L,B
[16]   →RET U
     ∇
```

Figure 1: Prime number generator

Application of a "Hybrid" (Two-Mode) Processing System

by

P.H. Russel, L. Wagschal, and J.C. Wiginton

ABSTRACT

The adjective "hybrid" is used in this context in its meaning "heterogeneous"; specifically, the combination of time-shared and batch processing to achieve an objective in an optimal manner. Such a system would have at least three generic desiderata. The first requirement would be a filing system whose files are accessible through both time-sharing and batch systems. The second requirement is a time-sharing system with the important features of allowing dynamic storage maintenance, and providing remote job entry capability. The third requirement is a batch processor which itself is efficient in resource utilization, and has the capacity of providing the time-sharing system with optimized input and output streams. Each of these requirements has been met in several applications, two of which are described.

The first application is a sophisticated simulations model, FINANSIM, used for operational teaching of financial management. The model was originally implemented in FORTRAN and, aside from extensive card-handling, was reasonably sophisticated and efficient. Complete reprogramming in available time-sharing languages was undesirable because of cost and time, and probable degradation of operating efficiency. On the other

hand, it was not desirable to have the student learn a new time-sharing language, FORTRAN, or OS/JCL. Card-handling was, however, to be totally eliminated, but the system should permit maximum input verification by users. Data entry was to be remote both for demonstration, and since the user group was not located at the main computer centre. Finally, the system was to provide each user with his own private FINANSIM history file, with controlled back-tracking capability. This extensive list of requirements was fully met using the York/APL time-sharing system for interactive data entry, error checking, RJE job submissions, and task monitoring of the time-shared and batch processing phases of operation at both micro and macro levels. The related York/APL File I/O system provides the means of making the data entered from terminals useable by the batch processor as well as segregating output between printing and the private history files. The batch processing is accomplished by the original FORTRAM program, with very minor modifications, under the HASP/OS monitor. The whole package operates on an IBM System 370/155.

The second application is a system for remote conferencing, CONCORD (conference coordinator), in which the participants develop problem-solving strategies. All interaction is through an information retrieval system which each conference participant may access. The system automatically maintains a log or protocol of each conference. On termination of each conference, a batch processing system is invoked which evaluates the conference data by statistical analysis of files to measure the efficiency or effectiveness of the conferees as problem-solvers for their benefit. This sophisticated system has been implemented using the York/APL File I/O system to share files between York/APL and HASP/OS. The participants communicate using keyboard terminals via the shared files. The batch processing uses readily available data base management and statistical analysis programs.

AN APL TRANSLATOR FOR THE B6700

I.M. Braude

M.A. Jenkins

Computing and Information Science
Queen's University

ABSTRACT

APL is a convenient notation for describing many algorithms of interest to scientists, engineers and mathematicians. Current implementations make exploitation of the notation to solve large, practical, computational problems quite expensive compared to batch processing languages. A translator which compiles a restricted version of APL into B6700 ALGOL has been developed for batch use of APL. The major difference from standard APL is that identifiers must always refer to values of the same type and rank; this restriction leads to some modifications to standard APL notation. The suitability of the restricted language and the design of the translator are discussed and timing results compared with APL/700.